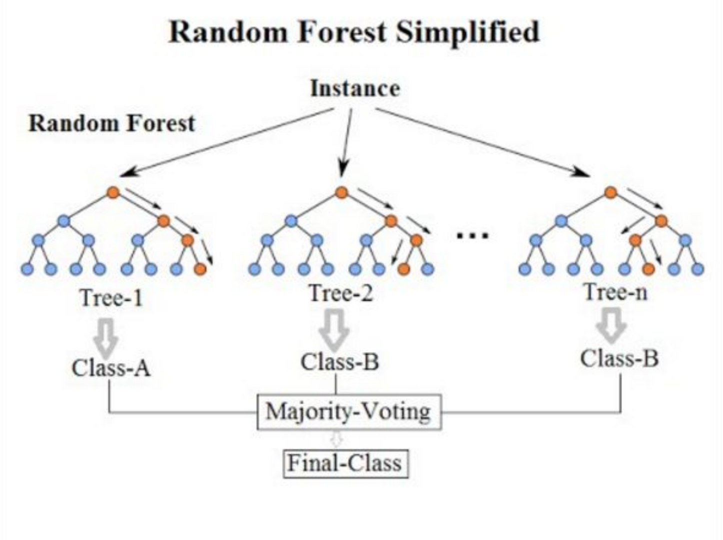
Random Forest
Final Presentation

Liying Chen, Mufei Chen,
Mandy Meng Ni Ho, Catherine Zhang

Introduction

Random Forest is a method for classification that operates by constructing **a multitude of decision trees** at training time and outputting the **mean prediction** of the individual trees.

Random Forest **randomly** constructs decision trees which are represented by subset matrices of the input matrix, then each subset matrix gives a prediction for classification. Mean prediction given by all of subset matrices is the final output given by random forest.



Pros and Cons

Pros:

- Less likely to overfit than single trees
- Less variance than single decision tree
- Simple to adjust the algorithm to different data sets
- Can predict what variables are important to the algorithm

Cons

- Can overfit for noisy datasets
- Computationally expensive: construction and prediction can be time-consuming
- Black box: Can be difficult to visualize the model

Simple Model

Should a given patient be worried about their back pain?

Back Pain Data

- Free online Kaggle dataset
- 100 healthy patients, 200 sick patients
- 12 different data attributes

Attribute1 = pelvic_incidence (numeric)
Attribute2 = pelvic_tilt (numeric)
Attribute3 = lumbar_lordosis_angle (numeric)
Attribute4 = sacral_slope (numeric)
Attribute5 = pelvic_radius (numeric)
Attribute6 = degree_spondylolisthesis (numeric)
Attribute7= pelvic_slope(numeric)
Attribute8= Direct_tilt(numeric)
Attribute9= thoracic_slope(numeric)
Attribute10= cervical_tilt(numeric)
Attribute11=sacrum_angle(numeric)
Attribute12= scoliosis_slope(numeric)

	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11	Col12	Class_att
63.0278175	22.55258597	39.60911701	40.47523153	98.67291675	-0.254399986	0.744503464	12.5661	14.5386	15.30468	-28.658501	43.5123	1
39.05695098	10.06099147	25.01537822	28.99595951	114.4054254	4.564258645	0.415185678	12.8874	17.5323	16.78486	-25.530607	16.1102	1
68.83202098	22.21848205	50.09219357	46.61353893	105.9851355	-3.530317314	0.474889164	26.8343	17.4861	16.65897	-29.031888	19.2221	1
69.29700807	24.65287791	44.31123813	44.64413017	101.8684951	11.21152344	0.369345264	23.5603	12.7074	11.42447	-30.470246	18.8329	1
49.71285934	9.652074879	28.317406	40.06078446	108.1687249	7.918500615	0.543360472	35.494	15.9546	8.87237	-16.378376	24.9171	1
43.11795103	13.81574355	40.34738779	29.30220748	128.5177217	0.970926407	0.110795865	8.9802	15.1873	10.59114	-17.943314	33.0483	0
40.6832291	9.148437195	31.02159252	31.53479191	139.1184721	-2.511618596	0.775688024	31.2682	13.6632	13.015	-4.591917	19.9869	0
37.7319919	9.386298276	41.99999999	28.34569362	135.740926	13.68304672	0.465169721	28.9703	10.2016	11.24951	-19.160909	34.0011	0
63.92947003	19.97109671	40.17704963	43.95837332	113.0659387	-11.05817866	0.412296214	19.7733	11.1443	7.97351	-7.809627	29.5091	0
61.82162717	13.59710457	63.99999999	48.22452261	121.779803	1.296191194	0.629660667	17.9906	13.6082	8.34518	-10.939434	20.7594	0
62.14080535	13.96097523	57.99999999	48.17983012	133.2818339	4.955105669	0.122419736	33.8766	16.3819	9.66244	-16.783645	43.8402	0
69.00491277	13.29178975	55.5701429	55.71312302	126.6116215	10.83201105	0.385073208	35.4534	7.4752	7.76405	-11.716465	13.0886	0
56.44702568	19.44449915	43.5778464	37.00252653	139.1896903	-1.859688529	0.076818991	11.2853	16.1623	14.93052	-12.656406	40.5705	0
41.6469159	8.835549101	36.03197484	32.8113668	116.5551679	-6.054537956	0.098119047	10.0549	8.7771	8.64451	-5.079724	29.4263	0
51.52935759	13.51784732	35	38.01151027	126.7185156	13.92833085	0.863545131	33.2628	11.087	12.42093	-15.259539	18.2936	0
39.08726449	5.536602477	26.93203835	33.55066201	131.5844199	-0.75946135	0.252511739	26.2684	12.6508	12.36056	-34.071611	43.7183	0
34.64992241	7.514782784	42.99999999	27.13513962	123.9877408	-4.082937601	0.419743489	29.72	15.279	16.49241	-3.437709	21.8868	0
40.25019968	13.92190658	25.1249496	26.32829311	130.3278713	2.230651729	0.789992856	29.323	12.0036	10.40462	-1.512209	9.6548	1
53.43292815	15.86433612	37.16593387	37.56859203	120.5675233	5.988550702	0.198919573	13.8514	10.7146	11.37832	-20.510434	25.9477	1
45.36675362	10.75561143	29.03834896	34.61114218	117.2700675	-10.67587083	0.131972555	28.8165	7.7676	7.60961	-25.111459	26.3543	1
43.79019026	13.5337531	42.69081398	30.25643716	125.0028927	13.28901817	0.190407626	22.7085	11.4234	10.59188	-20.020075	40.0276	1
36.68635286	5.010884121	41.9487509	31.67546874	84.24141517	0.664437117	0.367700139	26.2011	8.738	14.91416	-1.702097	21.432	1
49.70660953	13.04097405	31.33450009	36.66563548	108.6482654	-7.825985755	0.6880095	31.3502	16.5097	15.17645	-0.502127	18.3437	1
31.23238734	17.71581923	15.5	13.51656811	120.0553988	0.499751446	0.608342758	21.4356	9.2589	14.76412	-21.724559	36.4449	1
48.91555137	19.96455616	40.26379358	28.95099521	119.321358	8.028894629	0.139478165	32.7916	7.2049	8.61882	-1.215542	27.3713	1
53.5721702	20.46082824	33.1	33.11134196	110.9666978	7.044802938	0.081930993	15.058	12.8127	12.00109	-1.734117	15.6205	1
57.30022656	24.1888846	46.99999999	33.11134196	116.8065868	5.766946943	0.416721511	16.5158	18.6222	8.51898	-33.441303	13.2498	1
44.31890674	12.53799164	36.098763	31.78091509	124.1158358	5.415825143	0.664040876	9.5021	19.1756	7.25707	-32.893911	19.5695	1
63.83498162	20.36250706	54.55243367	43.47247456	112.3094915	-0.622526643	0.560675371	10.769	16.8116	11.41344	2.676002	17.3859	1
31.27601184	3.14466948	32.56299592	28.13134236	129.0114183	3.623020073	0.534481238	31.1641	18.6089	8.4402	4.482424	24.6513	1
38.69791243	13.44474904	31	25.25316339	123.1592507	1.429185758	0.30658054	28.3015	17.9575	14.75417	-14.252676	24.9361	1
41.72996308	12.25407408	30.12258646	29.475889	116.5857056	-1.244402488	0.468525928	28.5598	12.4637	14.1961	-20.392538	33.0265	1

Link: [Dataset_spine]

```
#GENERAL GUIDELINES ON THE STEPS TO RANDOM FOREST CODE

FIRST STEP
• import libraries
        - usually import pandas as pd, import numpy as np

SECOND STEP
• import data and create a dataset using .read_csv or a DataFrame object

THIRD STEP
• prepare data for training
        - separate into attributes and labels (dependent and indepnedent) using either iloc or DataFrame interface
        - split into test/training sets
                -> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
                -> X is the attributes, y is the labels

(OPTIONAL STEP for quantative data)
• scale the data
        from sklearn.preprocessing import StandardScaler

        sc = StandardScaler()
        X_train = sc.fit_transform(X_train)
        X_test = sc.transform(X_test)

FOURTH STEP
• train the algorithm
        from sklearn.ensemble import RandomForestClassifier

        classifier = RandomForestClassifier(n_estimators=20, random_state=0)
        classifier.fit(X_train, y_train)
        y_pred = classifier.predict(X_test)


FIFTH STEP
• evaluate the accuracy
        from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

        print(confusion_matrix(y_test, y_pred))
        print(classification_report(y_test, y_pred))
        print(accuracy_score(y_test, y_pred))
```

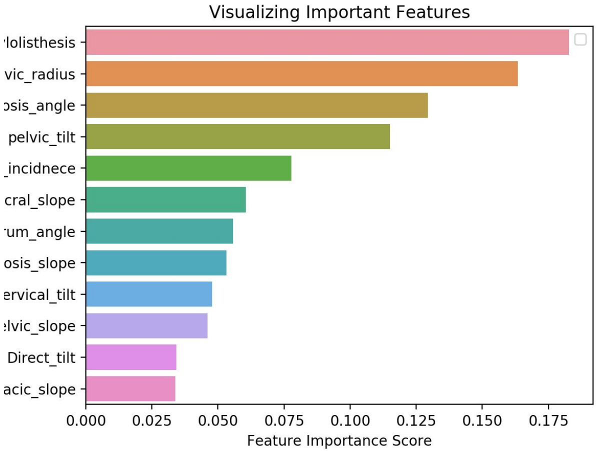
Link: [back_pain_random_forest_1.py]

Preprocessing Data

```
#divides data into training and testing sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)

# 4) feature scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# 5) training the algorithm
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n_estimators=25, random_state=1)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

# 6) Evaluating the algorithm
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print("Accuracy: ",metrics.accuracy_score(y_test, y_pred))

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test, y_pred))
```

Feature Importance

- Random forest provides a built in feature importance score
- The higher the feature importance score, the bigger impact the feature has on the resulting model




```
# **************** FINDING THE IMPORTANT FEATURES ********************************

# 1) Create a random forests model
# 2) Use the FEAUTRUE IMPORTANCE VARIABLE to see the feature importance scores
# 3) Visualize these scores using the SEABORN LIBRARY

col_name = ['pelvic_incidnece','pelvic_tilt','lumbar_lordosis_angle','sacral_slope','pelvic_radius','degree_spondylolisthesis',
            'pelvic_slope','Direct_tilt','thoracic_slope','cervical_tilt','sacrum_angle','scoliosis_slope']

feature_imp = pd.Series(classifier.feature_importances_,index=col_name).sort_values(ascending=False)
feature_imp



#visualizing the feature importance
import matplotlib.pyplot as plt
import seaborn as sns

#creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
#adding labels to the graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()
```

Results

- We remove the features with the lowest feature importance score, and improve our accuracy.

```
#************************* GENERATING MODEL WITH SELECTED FEATURES ****************************
# removed direct_tilt, thoracic_slope



X=df[['pelvic_incidnece','pelvic_tilt','lumbar_lordosis_angle','sacral_slope','pelvic_radius','de
lope','cervical_tilt','sacrum_angle','scoliosis_slope']]
y=df['Class_att']


print("SELECTED FEATURES")

print(df.head(10))


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.70, random_state=5)


#generating a model
clf=RandomForestClassifier(n_estimators=100)

clf.fit(X_train, y_train)

y_pred=clf.predict(X_test)

print("Accuracy: ",metrics.accuracy_score(y_test, y_pred))
```

```
Catherines-MacBook-Pro-2:mdp catzhang$ python back_pain_random_forest_1.py
   pelvic_incidnece  pelvic_tilt    ...      scoliosis_slope  Class_att
0         63.027818    22.552586    ...              43.5123          1
1         39.056951    10.060991    ...              16.1102          1
2         68.832021    22.218482    ...              19.2221          1
3         69.297008    24.652878    ...              18.8329          1
4         49.712859     9.652075    ...              24.9171          1
5         43.117951    13.815744    ...              33.0483          0
6         40.683229     9.148437    ...              19.9869          0
7         37.731992     9.386298    ...              34.0011          0
8         63.929470    19.971097    ...              29.5091          0
9         61.821627    13.597105    ...              20.7594          0

[10 rows x 13 columns]
('Accuracy: ', 0.7661290322580645)
[[25 19]
 [10 70]]
              precision    recall  f1-score   support

           0       0.71      0.57      0.63        44
           1       0.79      0.88      0.83        80

   micro avg       0.77      0.77      0.77       124
   macro avg       0.75      0.72      0.73       124
weighted avg       0.76      0.77      0.76       124

0.7661290322580645
SELECTED FEATURES
   pelvic_incidnece  pelvic_tilt    ...      scoliosis_slope  Class_att
0         63.027818    22.552586    ...              43.5123          1
1         39.056951    10.060991    ...              16.1102          1
2         68.832021    22.218482    ...              19.2221          1
3         69.297008    24.652878    ...              18.8329          1
4         49.712859     9.652075    ...              24.9171          1
5         43.117951    13.815744    ...              33.0483          0
6         40.683229     9.148437    ...              19.9869          0
7         37.731992     9.386298    ...              34.0011          0
8         63.929470    19.971097    ...              29.5091          0
9         61.821627    13.597105    ...              20.7594          0

[10 rows x 13 columns]
('Accuracy: ', 0.8248847926267281)
```

More Complex Model

Does this patient have Soft Tissue Sarcoma or Multiple Sclerosis?

Soft Tissue Sarcoma

- A rare type of cancer that begins in the tissues that connect, support and surround other body structures.
    - There are a lot of subtypes, so it takes time to differentiate subtypes and diagnose.
    - Even if patients have same disease, the outcome of disease varies
    - A common method to monitor responses after non-surgical treatments is Multi-parametric MRI, but it may not reveal all post treatment changes; it is also because STS tumors are diverse in features, including cell tumors, necrosis and tissue compartment.

Multiple Sclerosis

- A disease in which the immune system eats away at the protective covering of nerves.
  - Face a challenge that predict the individual patient evolution and responses to therapy

Suggestions for future study

- Instead of helping diagnose, machine learning study may focus on prediction of patient evolution in the future since multiple sclerosis cannot be cured at present.
- For both disease, predict post treatment reactions is helpful.

Method

- Replace missing data with medians to make the dataset easier to fit the model.
- Split dataset into training dataset and testing dataset.
  - Testing dataset is a dataset which provides classification results as a reference
  - training dataset is used for prediction.
- Fit the model on testing data to predict classification results of samples in training dataset.
- Results in mean predictions
  - Prediction for each person in training dataset either converges at 1 or 0, which means if the person has STS or multiple sclerosis
- Calculate accuracy.

Data Extraction

- Extract 5000 records for each disease state, after merging, there were 642 subjects
- Outcome: Two disease (from table `diagnoses`)
  - Soft tissue tumor = 1 (117 subjects)
  - Multiple sclerosis = 0 (525 subjects)

Imbalance issue??

- Features: lab results (from table `labresults`)
  - Albumin level
  - Calcium level
  - Cholesterol
  - Protein level
  - Hemoglobin

SQL code to extract data from the server

```
\copy (select a.encounterid, a.termnamemapped,  b.result_name, b.value
from diagnoses as a join labresults as b on a.encounterid = b.encounterid
where a.termnamemapped = 'Multiple sclerosis' and (b.result_name =
'ALBUMIN LEVEL' or  b.result_name = 'CALCIUM LEVEL' or  b.result_name =
'CHOLESTEROL' or b.result_name = 'PROTEIN LEVEL' or b.result_name =
'Hemoglobin') limit 5000 ) TO 'C:\Users\mandyho\Desktop\data_MS_5000.csv'
CSV HEADER;
```

```
machinelearning=> \copy (select a.encounterid, a.termnamemapped,  b.result_name, b.value from diagnoses as a join labresults as b on a.encounterid = b.encounterid where a.termna
memapped = 'Multiple sclerosis' and (b.result_name = 'ALBUMIN LEVEL' or b.result_name = 'CALCIUM LEVEL' or  b.result_name = 'CHOLESTEROL' or b.result_name = 'PROTEIN LEVEL' or b
.result_name = 'Hemoglobin') limit 5000) TO 'C:\Users\mandyho\Desktop\data_MS_5000.csv' CSV HEADER;
WARNING:  temporary file leak: File 45 still referenced
WARNING:  temporary file leak: File 114 still referenced
COPY 5000
machinelearning=> \copy (select a.encounterid, a.termnamemapped,  b.result_name, b.value from diagnoses as a join labresults as b on a.encounterid = b.encounterid where a.termna
memapped = 'Neop, mlig, soft tissue NOS' and (b.result_name = 'ALBUMIN LEVEL' or b.result_name = 'CALCIUM LEVEL' or  b.result_name = 'CHOLESTEROL' or b.result_name = 'PROTEIN LE
VEL' or b.result_name = 'Hemoglobin') limit 5000) TO 'C:\Users\mandyho\Desktop\data_ts_5000.csv' CSV HEADER;
WARNING:  temporary file leak: File 145 still referenced
WARNING:  temporary file leak: File 42 still referenced
COPY 5000
```

Data Manipulation

- Remove “Cholesterol”: no subjects contain value in cholesterol
- Missing value imputation: replace with median

```
>>> combine.isna().sum()
result
ALBUMIN LEVEL     118
CALCIUM LEVEL      46
CHOLESTEROL       621
Hemoglobin         91
PROTEIN LEVEL     120
outcome             0
dtype: int64
```

imputation

```
>>> combine_impute = pd.DataFrame(combine_impute)
>>> combine_impute.isna().sum()
0    0
1    0
2    0
3    0
4    0
dtype: int64
```

- Cleaned data:

```
>>> combine.head()
result                                              ALBUMIN LEVEL  CALCIUM LEVEL  Hemoglobin  PROTEIN LEVEL  outcome
idx
00EB5B299647EA98AA104BB287698605F59B8E4ABDBDA1F...            4.2            9.2        10.4            6.9        1
011702F949386E96DA8FD9CA212E26380E541D8DAF95D2B...            3.8            9.2        14.2            6.8        1
0126366EC46DB8BCAEE827E57B569A3CE6FA65506FB6BE0...            NaN            9.6        14.6            NaN        1
020E73022EB983CA3B3256D70C7C38502104188372445D7...            4.3            9.6        12.8            7.7        1
02708B98A4B878D936537EE98968CB42BF07161DFAC24EA...            3.7            9.4         8.0            7.0        1
```

Model Fitting

```
>>> print('Training Features Shape:', train_features.shape)
Training Features Shape: (449, 4)
>>> print('Training Labels Shape:', train_labels.shape)
Training Labels Shape: (449,)
>>> print('Testing Features Shape:', test_features.shape)
Testing Features Shape: (193, 4)
>>> print('Testing Labels Shape:', test_labels.shape)
Testing Labels Shape: (193,)
```

- Training and testing set
  - Train: 70% (449 subjects)
  - Test: 30% (193 subjects)
- Random forest:
  - Set trees = 500 (the model will be fitted 500 times until it gets the best result)
  - Python library sklearn: RandomForestClassifier

```
>>> model = RandomForestClassifier(n_estimators=500,
...                                bootstrap = True,
...                                max_features = 'sqrt')
>>> # Fit on training data
... model.fit(train_features, train_labels)
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='sqrt', 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=500, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
>>> predictions = model.predict(test_features)
```

Results 1: without Normalization

```
>>> results = confusion_matrix(test_labels, predictions)
>>> print('Confusion Matrix :')
Confusion Matrix :
>>> print(results)
[[  3  32]
 [ 14 144]]
>>> print('Accuracy Score :',accuracy_score(test_labels, prediction
Accuracy Score : 0.7616580310880829
>>> print('Report : ')
Report :
>>> print(classification_report(test_labels, predictions))
              precision    recall  f1-score   support

         0.0       0.18      0.09      0.12        35
         1.0       0.82      0.91      0.86       158

   micro avg       0.76      0.76      0.76       193
   macro avg       0.50      0.50      0.49       193
weighted avg       0.70      0.76      0.73       193
```

Accuracy: 76.17%

Precision = TP/TP+FP
Recall = TN/TN+FN
F1 score: precision* recall / (precision + recall)

```
>>> print(predictions)
[1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1.
 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1.]
>>> rf_probs = model.predict_proba(test_features)[:,1]
>>> print(rf_probs)
[0.942      0.976      0.976      0.18290476 0.93       0.956
 0.6612     0.926      0.982      0.812      0.628      0.988
 0.80083333 0.388      0.998      0.498      0.91       0.944
 0.94       0.966      0.336      0.712      0.964      0.94266667
 0.894      0.86716667 0.982      0.562      0.796      0.876
 0.538      0.74       0.92566667 0.81       0.79       0.2605
 0.59903333 0.97       0.848      0.964      0.952      0.926
 0.914      0.72       0.99733333 0.124      0.9215     0.76053333
 0.894      0.424      0.636      0.426      0.966      0.524
 0.81       1.         0.708      0.96733333 0.988      0.98266667
 0.918      0.624      0.822      0.954      0.85866667 0.794
 0.53       0.964      0.854      0.98       0.814      0.362
 1.         0.874      0.674      0.984
 0.558      0.864      0.946      0.818
 0.772      0.824      0.976      0.37
 0.852      0.37       0.888      0.76
 0.89       0.52866667 0.58       0.87
 0.946      0.97       0.764      0.976
 0.912      0.282      0.748      0.5751     0.872      0.982
 0.948      0.91316667 0.642      1.         0.95       0.744
 0.78       0.706      0.882      0.824      0.952      0.829
 0.79003333 0.87       0.64       0.734      0.814      0.874
 0.988      0.696      1.         0.666      0.876      0.388
 0.97       0.7502     0.598      0.872      0.656      0.976
 0.82133333 0.608      0.768      0.984      0.6985     0.56
 0.508      0.878      0.3809381  0.58       0.902      0.662
 0.69       0.604      0.768      0.444      0.622      0.696
 0.972      0.932      0.984      0.748      0.87617619 0.942
 0.832      0.932      0.97466667 0.946      0.822      0.882
 0.91       0.994      0.936      0.828      0.848      0.542
 0.69       0.974      0.92566667 0.986      0.932      0.99
 0.17566667 0.518      0.628      0.464      0.946      0.774
 0.624     ]
```

Classification result

Probability result
(p>0.5 -> y =1)

Result 2: Normalize the features

- Normalization: (x - mean(x))/std(x)

```
>>> features_new
          n0        n1        n2        n3
0   0.428452 -0.070993 -0.624860  0.532433
1  -0.629495 -0.070993  1.203625  0.365201
2   0.163965  0.728608  1.396097  0.030737
3   0.692939  0.728608  0.529973  1.870287
4  -0.893982  0.328808 -1.779692  0.699664
5   0.957426  0.528708  1.299861  0.866896
6   1.750886  0.528708  0.626209  0.866896
7   0.163965  0.128908  0.578091  0.030737
8   0.957426  2.327808  0.000675  0.866896
9  -0.365008  0.728608 -1.346630  0.365201
10 -0.365008 -0.470793  1.829159  0.532433
11 -0.100521  0.128908 -0.576742 -0.136494
12  0.163965  0.928508  0.241265  0.030737
13 -1.158468 -1.670193 -1.827810 -1.474349
14  0.163965  0.128908 -0.961686  0.030737
15  0.692939  0.328808  1.107389  0.699664
16 -0.100521 -0.670693  0.385619  0.365201
17 -1.158468 -1.470293 -0.672978  1.034128
18  0.163965 -0.070993 -0.239915 -0.136494
19 -0.365008  0.528708 -0.528624  0.197969
20  0.692939  0.328808  1.732923  0.365201
21 -0.100521 -0.470793  0.433737  0.365201
22  0.163965 -0.470793  0.000675  0.030737
23  0.163965  0.528708  0.000675  0.030737
```

```
>>> results = confusion_matrix(test_labels, predictions)
>>> print('Confusion Matrix :')
Confusion Matrix :
>>> print(results)
[[  3  32]
 [ 16 142]]
>>> print('Accuracy Score :',accuracy_score(test_labels, predictions) )
Accuracy Score : 0.7512953367875648
>>> print('Report : ')
Report :
>>> print(classification_report(test_labels, predictions))
              precision    recall  f1-score   support

         0.0       0.16      0.09      0.11        35
         1.0       0.82      0.90      0.86       158

   micro avg       0.75      0.75      0.75       193
   macro avg       0.49      0.49      0.48       193
weighted avg       0.70      0.75      0.72       193
```

Accuracy: 75.13%

Results

- Overfitting

Accuracy:
75.13%

```
>>> results = confusion_matrix(test_labels, predictions)
>>> print('Confusion Matrix :')
Confusion Matrix :
>>> print(results)
[[  4  31]
 [ 17 141]]
>>> print('Accuracy Score :',accuracy_score(test_labels, predictions) )
Accuracy Score : 0.7512953367875648
>>> print('Report : ')
Report :
>>> print(classification_report(test_labels, predictions))
              precision    recall  f1-score   support

         0.0       0.19      0.11      0.14        35
         1.0       0.82      0.89      0.85       158

   micro avg       0.75      0.75      0.75       193
   macro avg       0.51      0.50      0.50       193
weighted avg       0.71      0.75      0.73       193
```

```
>>> model = RandomForestClassifier(n_estimators=200,
...                                bootstrap = True,
...                                max_features = 'sqrt')
>>> model.fit(train_features, train_labels)
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='sqrt', 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=200, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

Limitations

Reason for the "Moderate" A

1. Among the 10,000 recor
   values which we need dr
   patients with 4 test-resu
   -- Increase the size of d
   -- For test categories, o
   harms to patients and e
   Also, we are suffering fr
2. The dataset is unbalance
   individuals suffered from
   predict the new patient a
   -- next time we will extr

```
>>> results = confusion_matrix(test_labels, predictions)
>>> print('Confusion Matrix :')
Confusion Matrix :
>>> print(results)
[[  3  32]
 [ 14 144]]
>>> print('Accuracy Score :',accuracy_score(test_labels, predic
Accuracy Score : 0.7616580310880829
>>> print('Report : ')
Report :
>>> print(classification_report(test_labels, predictions))
              precision    recall  f1-score   support

         0.0       0.18      0.09      0.12        35
         1.0       0.82      0.91      0.86       158

   micro avg       0.76      0.76      0.76       193
   macro avg       0.50      0.50      0.49       193
weighted avg       0.70      0.76      0.73       193
```

	ST_real	MS_rea
ST_pred	3	32
MS_pred	14	144

Limitations & recommendations

3. Used median, a coarse method, to deal with the missing data.

-- Try some other methods, such as KNN imputations or using algorithms which can handle missing data, ie Mixed Model.

- If the feature importance score for that feature is not important, then just remove it; or if very few people have that feature remove it.

4. Due to the lack of information for health people, we compares data of soft tissue disease and multiple sclerosis. This may decrease the accuracy because of the associations between diseases, leading to similarity of some test results.

-- **Finding data from other data sets to supplement our data set**

- hard to find data with same features as extracted from the database, since the

.

Conclusion

We established a process of extracting data, variables pre-selection, data pre-processing (ie: coding string and missing data) and model building. Though the accuracy is moderate, largely due the structure of dataset, our framework might inspire the members next year.

Thank you!!