Liying Chen
Mufei Chen
Mandy Meng Ni Ho
Catherine Zhang

MDP: Machine Learning with Biometrics Data for Personal Health Diagnosis

Random Forest Subteam Final Report

Link to Google Drive : https://drive.google.com/open?id=1rdwm1Rq2w5mSfKJyiNJYyyhD4TJ-0mLz

I. Executive Summary

"Big Data" can be used to predict health but are very large datasets that are difficult to interpret in meaningful ways with basic statistical methods. Personal health sensors and administrative medical records create Big Data that can be organized and categorized by machine learning techniques to enhance patient and clinician understanding and detection of early stage disease. Students on this project will use health data (such as EKGs, pulse, tissue parameterization, and brain waves) to develop machine learning based algorithms to translate this data into meaningful health improvement.

II. Team Objectives

This team will use machine learning techniques to analyze MGI Big Data. In the future, we aim to collect and analyze data from wearable sensors, cell phone health apps, and other sources for collecting real time biometric data. Our team focuses specifically on the Random Forest algorithm and the applications of Random Forest to predicting diseases and to making differential diagnosis based on these collected data.

Differential diagnosis[1] is an important part in clinic. It aims to distinguish a particular disease or condition from others that present similar clinical features. We will look at how basic lab results, including Albumin level, Calcium level, Cholesterol, Protein level, Hemoglobin can be used in conjunction with the Random Forest model to diagnose whether patients with certain lab results are more likely to have multiple sclerosis or soft tissue tumor.

We hope that our results will help to future differential diagnosis and finally achieved the final objectives of predicting disease.

III. Methods

Random Forest[2] 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. Term "Random Forest" can be considered as a collection of decorrelated decision trees and represented by a matrix. Rows of the matrix are samples in the dataset which students apply algorithms to, and columns are different features of these samples. 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.

- Resources to help code random forest
  Python:
https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76
  R: https://uc-r.github.io/random_forests

Following are the step to extract data from the DataDirect Precision Health database using YRBC server:

- **Data Extraction**

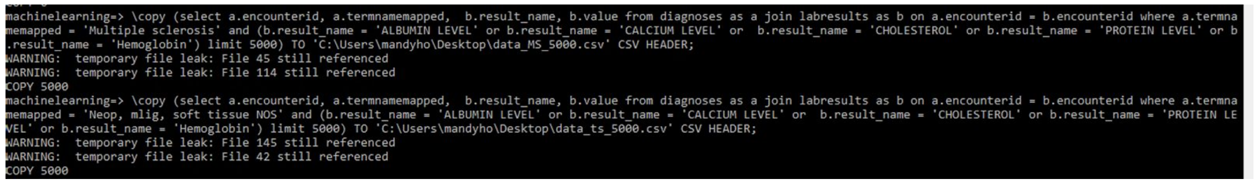
Our data is provided by Data Direct (https://datadirect.precisionhealth.umich.edu) from the Precision Health group of University of Michigan (https://precisionhealth.umich.edu) and is extracted using YRBC server. We performed SQL on the server that extract the data to CSV.
Following are the steps to access Armis2 server then extract the model

1. Login to YRBC virtual machine
2. Access to Machine Learning PostgreSQL Database
   ```
   psql -h 10.242.18.215 -d machinelearning -U uniqname
   ```
3. Perform SQL and extract data to CSV, data will be stored in the working directory on the Virtual Machine

```
\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\uniqname\Desktop\data_MS_5000.csv' CSV HEADER;

\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 = '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 LEVEL' or b.result_name =
'Hemoglobin') limit 5000) TO 'C:\Users\uniquname\Desktop\data_TS_5000.csv' CSV
HEADER;
```

In this SQL step, we basically joined the data from table `diagnosis` and table `labresults` by `patientid`; select two types of disease: multiple sclerosis and

soft tissue tumor of their calcium level, protein level, cholesterol, albumin level and hemoglobin as covariates.
Snapshot from the server if data extraction is performed correctly:



```
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 Explorations**

Variables	Codebook
`patientid`	Character, id for patients
`encounterid`	Character, id for visits of patients
`termnamemapped`	Character, disease name
`result_name`	Character, lab result name
`value`	Character, lab result value

- **Data preprocessing and Model Fitting:**

We will use library `pandas`[3] to do data cleaning and `sklearn`[4] for model fitting: our original variables in the data are `patientid`, `termnamemapped`, `result_name`, `value`, our outcome will be `termnamemapped` (multiple sclerosis and soft tissue tumor) and predictors will be `result_name`, `value`.

Step 1: Read in data

```
import pandas as pd
ms = pd.read_csv(r'C:\Users\mandyho\Desktop\data_MS_5000.csv')
ts = pd.read_csv(r'C:\Users\mandyho\Desktop\data_TS_5000.csv')
```

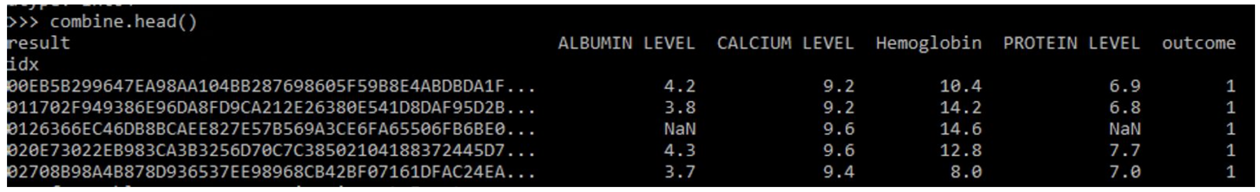
Step 2: Data processing

1. Change data from long to wide format, we are converting the four lab tests under `result_name` to five columns (i.e. covariates). Our covariates will be lab results such as calcium level, protein level, albumin level, cholesterol and hemoglobin, which are under `result_name`, and the covariates value is under `values`.
2. Convert `termnamemapped` to dummy variables, which `termnamemapped = 'Multiple sclerosis'` will be coded as 1, `termnamemapped = 'Neop, mlig, soft tissue NOS'` will be coded as 0.

```
# data processing: change long to wide
```

```
ts['idx'] = ts['encounterid']
ts['result'] = ts["result_name"]
ts_temp = ts.groupby(['encounterid', 'result_name']).first()
ts_wide = ts_temp.pivot(index = 'idx', columns = 'result', values = 'value')

ms['idx'] = ms['encounterid']
ms['result'] = ms["result_name"]
ms_temp = ms.groupby(['encounterid', 'result_name']).first()
ms_wide = ms_temp.pivot(index = 'idx', columns = 'result', values = 'value')

# add outcome
ts_wide['outcome'] = 1
ms_wide['outcome'] = 0

# combine
frame = [ms_wide, ts_wide]
combine = pd.concat(frame)

# check dimension
combine.shape
ms_wide.shape
ts_wide.shape
```

Snapshot of the 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
```

Step 3: Imputation of missing values, since Random forest in `sklearn` cannot take in any missing values, we use median to replace missing values. Moreover, we also drop covariates `cholesterol`, since all the value under this covariate is missing.

```
# check missing value
combine.isna().sum()

# drop cholesterol, contain too many missing value
combine = combine.drop("CHOLESTEROL", axis=1)

# missing values
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values='NaN', strategy='median', axis=0)
imputer = imputer.fit(combine)
combine_impute = imputer.transform(combine)
```

Step 4: Split the data to training and testing set using `train_test_split`
`labels` will be the matrix only contains outcome variables.
`feature_drop` will be the matrix contains all the predictors.

```
# Labels are outcome
labels = combine_impute[:,4]
# Remove the labels as features data
features = combine_impute[:,0:4]

# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
train_features, test_features, train_labels, test_labels =
train_test_split(features, labels, test_size = 0.4, random_state = 42)

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

Step 5: train the model using `RandomForestClassifier`, and set number of trees to 500.

```
from sklearn.ensemble import RandomForestClassifier

# Create the model with 100 trees
model = RandomForestClassifier(n_estimators=500,
                               bootstrap = True,
                               max_features = 'sqrt')
# Fit on training data
model.fit(train_features, train_labels)

# Import the model we are using
# from sklearn.ensemble import RandomForestRegressor
# Instantiate model with 500 decision trees
# rf = RandomForestRegressor(n_estimators = 500, random_state = 42)
# Train the model on training data
# rf.fit(train_features, train_labels)
```

Step 6: Fit the model on testing set using `model.predict`, calculate the predicted probability then print out the confusion matrix (accuracy)

```
# Use the forest's predict method on the test data
predictions = model.predict(test_features)
print(predictions)
rf_probs = model.predict_proba(test_features)[:,1]
print(rf_probs)

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

results = confusion_matrix(test_labels, predictions)
print('Confusion Matrix :')
print(results)
print('Accuracy Score :',accuracy_score(test_labels, predictions) )
print('Report : ')
print(classification_report(test_labels, predictions))
```

Step 7: Retrain the model with normalize data, set tree to 500
Since our accuracy for the model performed in step 6 is around 76%, we normalize the data by subtracting the mean and divided by the standard deviation, and redo step 4 to 6 see if accuracy increases.

```
# normalization
features_df['n0'] = (features_df.iloc[:,0] -
features_df.iloc[:,0].mean())/features_df.iloc[:,0].std()
features_df['n1'] = (features_df.iloc[:,1] -
features_df.iloc[:,1].mean())/features_df.iloc[:,1].std()
features_df['n2'] = (features_df.iloc[:,2] -
features_df.iloc[:,2].mean())/features_df.iloc[:,2].std()
features_df['n3'] = (features_df.iloc[:,3] -
features_df.iloc[:,3].mean())/features_df.iloc[:,3].std()

features_new = features_df.iloc[:,4:]
```

Step 8: Retrain the model with normalize data, set tree to 200

IV. Results and Discussion

In this section we will compare accuracy and confusion matrix component of three random forest model: (1) use raw data, set trees to 500 (2) Use normalized data, set trees to 500 (3) use normalized data, set trees to 200.

Accuracy is calculated by the sum of the total correct labels and divided with sum of all labels. ((TP+FP)/(TP+FP+FP+FN)). We can see that model using raw data and set trees to 500 has higher accuracy. In the confusion matrix, precision is the sum of true positive divided by sum of predicted positives (TP/(TP+FP))[5], for example, precision = 0.82 means we are having more predicted positive than true positive, this might be due to overfitting of the data. By looking at the below table, our precision for soft tissue tumor (disease = 1) is higher than multiple sclerosis (disease = 0). Next, recall is the ratio of true negative and predicted negative (TP/TP+FN)[5], and the recall for soft tissue tumor (disease = 1) is higher than multiple sclerosis (disease = 0). Finally, F1 score is precision* recall / (precision + recall), the higher the better. Therefore, by looking at the confusion matrix result, the model is better at predicting soft tissue tumor than multiple sclerosis.

Model	Accuracy	Recall	Precision	F1 score
Use raw data, set trees to 500	76.17%	0: 0.18 1: 0.82	0: 0.09 1: 0.91	0: 0.12 1: 1.86
Use normalized data, set trees to 500	75.13%	0: 0.16 1: 0.82	0: 0.09 1: 0.90	0: 0.11 1: 0.86
Use normalized data, set trees to 200	75.13%	0: 0.19 1: 0.82	0: 0.11 1: 0.89	0: 0.14 1: 0.85

V. Limitations and Recommendations

Because our data is obtained from patients who go to a hospital to have issues diagnosed, the data is and our available datasets include very few healthy people and a lot of sick people. Since it would be hard to accurately train the model to identify sick and healthy people when there is no data available on healthy people, we decided to create a model that looked at a sick patient's data and determined which disease (out of two diseases) a patient was more likely to have. This may help physicians with differential diagnosis, which is a vital part in the clinical medicine. Our current model compares data of soft tissue disease and multiple sclerosis and categorizes patients as one or either.

To be specific, the model cannot be built unless we have two categories to compare, either patients and healthy people or people with two different diseases. Technically, compare data from people with a specific disease to data obtained from healthy people will get a higher accuracy. The main reason for that phenomenon is the potential links between different diseases, while the connection between healthy people and patients appears to be weaker. Moreover, since we do not have enough data from healthy people, results will be imbalanced for the reason that even when we reject or accept all, we will still have a good accuracy. Hence we picked two diseases, soft tissue disease and multiple sclerosis, and compare them. The accuracy is about 76%.

There are a few reasons why this method has negative influence on the accuracy. First, only a few general lab tests are applied. Although soft tissue disease and multiple sclerosis are completely different diseases, some of their subtypes may share similar observations on these lab tests, so our computer feels it is hard to tell which disease the person exactly has while giving results. Second, it is hard to diagnose soft tissue disease. This feature makes machine learning a good tool for diagnosis, but soft tissue disease cannot be diagnosed by a simple model. Soft tissue disease has a few subtypes and different evolution tracks, so it cannot be defined by a general lab tests. In addition, soft tissue disease is one kind of tumor disease, so a dataset with pictures rather than statistical data is better for diagnosis.

A potential route that future teams can take is on the effectiveness and post-treatment evolution of multiple sclerosis. Since multiple sclerosis is an autoimmune disease and cannot be cured, it may be more beneficial to focus on predicting how the disease will change over time rather than on what merits a diagnosis of multiple sclerosis. However, if the research aims (of diagnosis) remain unchanged, then future researchers should work on finding data from other data sets to supplement our data set using data from emergency rooms. People typically go to emergency rooms for injuries and accidents, but most do not have diseases. However, hospitals may still take lab data, which we can use to supplement our data sets with data of non-diseased patients. A downside to exploring this route is that different emergency rooms and hospitals may take different types of data, which may be tedious to find and organize. Another potential solution would be to continue to expand the model with many more diseases.

Another limitation that we faced was that there was a lot of missing data. For example, for some of the columns that held numerical value data, some patients would have the entry “n/a,” indicating that this particular lab test was not performed on this patient. This is problematic because unlike other machine learning models, random forests requires each entry of the data to be filled with the same type of value, and having empty entries causes errors in the model. A possible option is just to exclude any features that have missing data, or exclude the patient who is missing the data. However, this would severely limit the amount of usable features in our model, and may lead to us missing crucial predictors for diseases and overfit the model to the features we can use.

Due to our team’s limited resources on the matter, we dealt with this problem by replacing all missing values with the median using the imputer method from the python sklearn package. However, this also leads to inaccurate predictions, as the model may interpret any value that deviates greatly from our calculated median to be abnormal (or normal if we are predicting a disease), and may identify that value as an important predictor of disease when in reality it is just a dummy value. A possible remedy for this in the future is to run the model using the dummy values and then determine the feature importance score for each feature involved. If the calculated feature importance score is not high, then we can assume that it is safe to exclude the feature. However, if the feature importance score is low, this does not exclude the possibility that this feature can be an important predictor if given all the right values. Future researchers may also want to look into the possibility of running a different algorithm that does not need complete data to function (such as KNN) to first select the important features, and then use these features to train the random forest model.

Regardless of which steps future researchers decide to take, we believe that a code-book detailing the types of data and significance of the data in the dataset is crucial to the team's future success. We will need these notes to decide how to pre-treat the data and modify the csv such that it accurately represents the data in the eyes of our model. For example, imagine a doctor classifies a symptom as type A, B, or C. In our data, they would be encoded as "A," "B," and "C." When we run our model on this data, the model will interpret this input as a range of numerical values, and predict accordingly. However, the data is not meant to be used as numerical values; rather, they are indicator variables. Thus, we would want to discard the original column and create three new columns instead, representing type A, B, and C, respectively. If a patient was categorized as type A, we would mark a 1 in the column representing A, and 0 in the columns representing B and C, so on and so forth. Another example in which we would need the codebook is to know the possible value from the data, especially for categorical variables, for example, if doctors are collecting level of proteins as "high", "low", "normal"; however, if we extract part of the data from the database and the value under protein level contains only "high" and "low", without the codebook telling us that there are actually three levels under protein level, based on the extracted data, we would never know there should also be a level "normal" under protein level. For these reasons, a code-book is a high priority.

Accomplishments and Contributions
Students start with kidney disease, and build a model for it. First, students replace missing data with medians to make the dataset easier to fit the model. Then, the dataset is split into training dataset and testing dataset. Testing dataset is a dataset which provides classification results as a reference to the model while running random forest method, and training dataset is used for prediction. After a random forest model is built by training data, students fit the model on testing data to predict classification results of samples in training dataset. Mean prediction for each person in training dataset converges at 1 or 0, which means if a person has a certain disease or not. Finally, results given by the model is compared to original results students have in training dataset to test the accuracy of the random forest model.

Individual Contributions
Liying Chen:
- Researched for Random Forest algorithm and found the related package for both R and python.
- Tuned the parameter and built random forest model via R 'caret' package in the toy example of Random forest using kidney data from UCI. Using ROC plot to show the accuracy.
- Explored and applied the method to extract data from both YRBC and ARMIS2.
- Generate python script for data preprocessing including replacing missing values, data cleaning(removed duplicates), and format changing (long to wide and dummy variables) and normalization
- Presentation: Limitation and Recommendation, Reference
- Final report: Team objectives, Limitation and Recommendation

Mufei Chen:
- Researched for Random forest algorithm (math part)
- Researched for specific disease that random forest method applied to
- Helped deal with missing data on kidney model.
- Contribute to extract data on Armis2
- Presentation: Soft tissue disease and multiple sclerosis
- Final Report: Method, Limitation and Recommendation, Accomplishments and Contributions

Mandy Meng Ni Ho:
- Helped with finding tutorials for random forest using python sklearn and pytorch.

- Helped with creating random forest toy example on chronic kidney disease data using python sklearn package.
- Contribute to extract data from the DataDirect server (YRBC and ARMIS2) and perform model random forest model fitting on Anaconda Python using *pandas* and *sklearn* libraries.
- Generate python script for data preprocessing (missing value imputation, data normalization, data transpose from long to wide) and model fitting.
- Presentation: Model building, Results
- Final report: Method, Results and Discussion, Limitation and Recommendation.

Catherine Zhang:

- Presented on Random Forest model and sklearn hyperparameters to teach team members about the model
- Coded and fine-tuned parameters for model predicting back pain using python sklearn libraries and functions
- Attended weekly meetings and managed communications between subteam and professor
- Researched and read papers on the use of Random Forest Models in bioinformatic fields and how to build good models
- Presentation: Introduction, Sample data set, General structure of Random Forest Model code, Simple model python code, Feature Importance Score
- Final Report: Executive Summary, Team Objectives, Limitations and Recommendations

VI. Reference

[1] Miller, D. H., et al. "Differential diagnosis of suspected multiple sclerosis: a consensus approach." Multiple Sclerosis Journal 14.9 (2008): 1157-1174.
[2] Liaw, Andy, and Matthew Wiener. "Classification and regression by randomForest." R news 2.3 (2002): 18-22.
[3] Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010) (publisher link)
[4] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[5] Confusion Matrix, Encyclopedia of Machine Learning and Data Mining, Ting, Kai Ming, 2017, 978-1-4899-7687-1, p.260