CS 536 : Final Project - Data Completion and Interpolation

Group members:

Name	netID	Workload
Xueyu WU	xw318	Analyze and clean data. Implement Random Forest model. Analyze the performance of RF and its working principles. Write related reports.
Yan GU	yg369	Data Analysis. Process nature language. Write reports.
Yunfan LI	yl1269	Data Analysis. Implement and analyze KNN algorithm. Write related reports.

1. Data Analysis and Cleaning

Our data set is ML3. Since it is an psychological questionnaire, there are so many questions, which are features and variables in our model, can reflect the personal character of different people.

1.1. Data Analysis

Basically, there are 2 kinds of variables. One is discrete questions. Generally speaking, these questions can be answered by selecting different options. For example, like the figure shown below, answers are discrete from ***1 Never or almost never true of me***, ***2 Sometimes true of me*** to ***5 Decline to answer***. Although there are some connections between answers, we still could not say they are contiguous because like, ***Sometimes true of me*** is definitely not twice as ***Never or almost never true of me***. So we still should set answers as discrete variables. Now we need classification models to make a prediction for each feature.

48	intrinsic_01	intrins	Individ	I enjoy tackling problems that are completely new to me.	Never or almost never true of me; Sometimes true of me;
49	intrinsic_02	intrins	Individ	I enjoy trying to solve complex problems.	Never or almost never true of me; Sometimes true of me;
50	intrinsic_03	intrins	Individ	The more difficult the problem, the more I enjoy trying to solve	Never or almost never true of me; Sometimes true of me;
51	intrinsic_04	intrins	Individ	I want my work to provide me with opportunities for increasing	Never or almost never true of me; Sometimes true of me;
52	intrinsic_05	intrins	Individ	Curiosity is the driving force behind much of what I do.	Never or almost never true of me; Sometimes true of me;

The other answer is open response. These features are consisted by nature languages, including words, number and sentences. In order to interpret the meaning of these variables, we need Nature Language Process. After that, we can not only use these features to classify other features, but predict open response answers.

1.2. Data cleaning

After observing raw data, there are something we need to do.

Firstly, there are some data whose all features are NA. We can not predict or classify based on nothing. So we delete these data at first.

Besides, different discrete variables have different ranges. For example, **mcdv1** has 7 classes from -3 to3. Features **mcfilter1** and **mcfilter2** have 4 and 5 classes respectively. However, **mcfilter1** is from 1 to 4 and **mcfilter2** from 0 to 4. So we need to scale all features in order to process them easier in our models.

BT	BU	BV	BW
mcdv1	mcdv2	mcfiller1	mcfiller2
NA	NA	NA	NA
NA	NA	NA	NA
0	3	1	0
0	2	1	0
3	3	2	1
0	3	1	0
0	2	2	1
-2	1	2	4
NA	NA	NA	NA
1	2	2	0
-1	-1	1	0
0	-1	2	0
1	2	2	0
0	2	2	0
0	2	1	0

2. Random Forest

After analyzing data, we can see there are some blanks. Take question *Most women are better off at home taking care of the children* as an example. If we want to predict the answers of it, we need other questions as variables.

However, we are facing two problems. Firstly, there are too many variables and most of them are irrelevant. Besides, even if we can prune some irrelevant variables, it could happened that some of these relevant variables are blank.

In order to solve these problems, we take random forest as one of our model for prediction. Decision trees are a specific way of expressing a classification function. Random forest is a bunch of decision trees trained by random variables and random data. So for discrete questions like Most women are better off at home taking care of the children, random forest can not only answer the question accurately, but ignore the impact of blanks.

a. Pruning variables
   Firstly, we prune irrelevant features. Based on ID3 Algorithm, features X with more information about Y can greatly attribute to our trees.

$$H(Y) = -\sum_{y} P(Y = y)\log\big(P(Y = y)\big)$$

$$H(Y|X) = \sum_{x} P(X = x)\,\mathrm{H}(Y|X = x)$$

$$= \sum_{x} P(X = x)\left(-\sum_{y} P(Y = y|X = x)\log\big(P(Y = y|X = x)\big)\right)$$

$$IG(X) = H(Y) - H(Y|X)$$

   So since there are over 200 variables and just few of them are relevant about our questions, we can prune variables at the first place and reduce the number of variables tremendously.
b. Picking randomly

Random Forest adds additional randomness to the model while growing trees. After searching for the most important features, we search for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model. Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. In addition, we pick training data also randomly. All these result in a more general and better classifier.

c. Building Decision Tree

Based on ID3 Algorithm, we can build our decision trees with selected features by greedy method.

1. For each randomly selected feature X_i, we compute $IG(X_i)$
2. Choose the feature with the maximal information gain
3. Split data based on this feature
4. Recursively apply this method on each part of the data, until all data is capture by the terminal node of some branch on the resulting decision tree

d. Bagging

Considering the result of decision trees, we can conclude that our classification returned by a decision tree is

$$F(x) = f(x)$$

However, one single model is going to have some weaknesses. And overfitting to training data is more likely to happen.

So by considering combining multiple trees together, the weaknesses of any one can be made up for in strengths of the others. After randomly pick training data and features, the strength of a forest is in diversity. Besides, a single tree can be trained very easily, and if we could then combine these simple trees to build the highly accurate complex models – Random Forest, we might be able to achieve the same performance more economically.

$$F(x) = majority(f_1(x_1) \dots f_N(x_N))$$

2.1. Construction

e. Data set

In order to discover the characteristic of each person, I select *big05* as my features. And keep the number of classes so that I can build corresponding brunch for them.

AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ
big5_01	big5_02	big5_03	big5_04	big5_05	big5_06	big5_07	big5_08	big5_09	big5_10
NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
5	3	6	5	5	3	5	1	5	5
3	5	7	7	2	7	2	1	1	7

In addition, since I want the value of each feature represent the index of the corresponding brunch, I need to -1 to make the value be from 0 to 6.

```
data = data - 1
```

a. Random feature

One advantage of random forests is that we can ignore some features when we make a classification. So for each tree, we need to select random features, and $random.sample()$ function is a good way to pick non-repetitive features.

```
feature = random.sample(self.head, self.d)
```

After picking features, we can select data whose these features are filled. Since our data has a lot of blanks, so choosing data according to these features are also random.

b. Data structure

In the previous assignments, we have solved problems about binary classification. We used a list to store our tree. However, now we are facing multiple classification questions. So it is necessary to build a data structure to preserve our tree. Now we are using key to remember the split point and a list as our children so that we can save all their brunches. As we mentioned, the index of each child represents the corresponding class.

```
class TreeNode(object):
    def __init__(self, key, num):
        self.key = key
        self.leaf = False
        self.num = num
        self.result = -1
        self.children = []
```

After a tree, we save the root node into our forest list. Then similarly, we randomly pick features and training data to build another tree until the forest is full.

```
    def build_forest(self):
        for i in range(self.m):
            data = pd.DataFrame()
            while data.empty:
                feature = random.sample(self.head, self.d)
                idx = feature + [self.goal]
                data = self.train_data[idx].dropna(subset=feature)
            root = self.build_decision_tree(data)
            self.forest.append(root)
            print('No.', i, 'Decision Tree built successfully!')
        print('Random Forest built successfully!')
```

c. Prediction

Now we have a forest. For each tree, we go down from root to leaf according to the value of our test data. If there is no data for the current key, this tree cannot classify our data. If we hit leaf, the result is our predicted class. After traverse all trees, the majority of our predicted class is the final classification.

2.2. Evaluation

a. Training error

Since this is a classification model, so for each data, the error is

$$\text{error} = \mathbf{1}\{\text{pred!} = \text{class}\}$$

And the overall average error is

$$\text{error} = \frac{1}{m}\sum \mathbf{1}\{\text{pred!} = \text{class}\}$$

Now we set the number of trees is 101. We can see that the training error is 0.2, correspondingly correctness is about 80%. For every feature we predicted, they all have 7 different classes, so the correctness of purely guessing is about $\frac{1}{7} \approx 14\%$. It is clear that our model does classify data into right class, which also means we can predict unfilled features correctly at some level.

```
No. 99 Decision Tree built successfully!
No. 100 Decision Tree built successfully!
Random Forest built successfully!
Train error:  0.20645491803278687
```

b. Testing error

Based on our *big05* features, the testing error is shown below.

```
Random Forest built successfully!
Train error:  0.10553278688524591
Test error:  0.6482617586912065
```

Although they have 7 classes and the correctness should be low at some extend, the test error is still surprised high. So there are 2 possible explanations. First is that these data are biased so that we cannot conclude one feature based on others. The other one is these features are irrelevant at some extend, so the rest of features can provide us no information to classify one feature.

In order to verify our conjectures, I use another set of features.

64	mcfiller1	mcfiller	Effect	Are you at all familiar with the building industry (trainings, relatives, ...)?
65	mcfiller2	mcfiller	Effect	Given your interests, how likely is it that you should be working in the building in
66	mcfiller3	mcfiller	Effect	Have you ever taken a recruiting decision in your life?
67	mcmost1	mcmost	Effect	Most women are better off at home taking care of the children.
68	mcmost2	mcmost	Effect	Men are more emotionally suited for politics than are most women.
69	mcmost3	mcmost	Effect	The best job for most women is something like cook, nurse, or teacher.
70	mcmost4	mcmost	Effect	Most women need a man to protect them.
71	mcmost5	mcmost	Effect	Most women are not really smart.
72	mcsome1	mcsome	Effect	Some women are better off at home taking care of the children.
73	mcsome2	mcsome	Effect	Men are more emotionally suited for politics than are some women.
74	mcsome3	mcsome	Effect	The best job for some women is something like cook, nurse, or teacher.
75	mcsome4	mcsome	Effect	Some women need a man to protect them.
76	mcsome5	mcsome	Effect	Some women are not really smart.
77	mcdv1	moninvignette	Effect	Do you feel that this job is better suited for one gender rather than the other?
78	mcdv2	moninvignette	Effect	In general, to what extent do you agree with the statement that women are just a

This data set are consisted with features with 2 classes, 4,5 classes and 7 classes. And after look through the description, we can conclude that all features are about women's characteristic and their career. They are strongly connected. I select *mcdv1, a 7 classes feature,* as our goal feature and after building a forest and evaluating our model, the error are shown below

```
branch = {'mcfiller1': 4, 'mcfiller2': 5, 'mcfiller3': 4,
          'mcmost1': 2, 'mcmost2': 2, 'mcmost3': 2, 'mcmost4': 2, 'mcmost5': 2,
          'mcsome1': 2, 'mcsome2': 2, 'mcsome3': 2, 'mcsome4': 2, 'mcsome5': 2,
          'mcdv1': 7, 'mcdv2': 7}
goal = 'mcdv1'
Random Forest built successfully!
C:/Users/Xueyu/PycharmProjects/phy/rf.py:145: DeprecationWarning: The truth valu
  if row[self.goal] != prediction:
Train error:  0.08632004038364463
Test error:  0.2217741935483871
```

As figure shown, the training error is 8% and testing error is 20%, which are pretty good considering our previous feature set. Now we can conclude that the reason why the previous one performs bed is that their features are less relevant to each other. So the key of our random forest model is to choose relevant features so that one can provide information to another.

c. Overfitting

Generally speaking, although there are gaps between training error and testing error, random forests are not likely overfitted. Because randomly picked features can ignore some irrelevant variables so that our decision is less affected by these details.

2.3. Analysis

a. Advantages

Firstly, random forest can deal with unfilled features very well. We can predict class even there is a few of features are valid. Besides, comparing to decision tree, forest can ignore

some irrelevant features without prone and is less likely overfitted. Finally, most questionnaires are consisted by questions which can be answered by options. So it is perfect for classification models.

b. Disadvantages
However, there are some disadvantages. Random forest can only deal with discrete variables. So we neither can get information from contiguous variables nor predict them. Besides, random forest can only classify one feature in one run. Although we can rewrite program to predict a set of features once, the number of brunch will explode so that we can hardly get a accurate model.

c. Improvements
For other classification model like k-means, they can classify all features at one time. Also they have different advantages which may make up the short comes of random forest. So a hybrid model may perform better.
Besides, models like autoencoder can not only classify discrete features as well as contiguous variables. So it is worthy to implement an autoencoder model in the future to make our prediction more accurate.

3. NLP

Considering the nature language processing, there are many methods we should use.

a. Edit distance
For two strings, the larger the edit distance, the more difference between them. So we can know how much difference is between the template word and the answer. For example, in question **anagrams 1-4**, they are just brainteasers. So, we can use the edit distance to tell the difference.

b. TF-IDF
In information retrieval, tf–idf or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. Tf–idf is one of the most popular term-weighting schemes today; 83% of text-based recommender systems in digital libraries use tf–idf.
The tf–idf is the product of two statistics, term frequency and inverse document frequency. There are various ways for determining the exact values of both statistics.
In the case of the term frequency $\mathrm{tf}(t, d)$, the simplest choice is to use the raw count of a term in a document, i.e., the number of times that term *t* occurs in document *d*. If we denote the raw count by $f_{t,d}$, then the simplest tf scheme is $\mathrm{tf}(t, d) = f_{t,d}$.

$$\mathrm{tf}(t, d) = 0.5 + 0.5 \times \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

The inverse document frequency is a measure of how much information the word provides, i.e., if it's common or rare across all documents. It is the logarithmically scaled inverse

fraction of the documents that contain the word (obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient):

$$\mathrm{idf(t, D)} = \log \frac{N}{|\{d \in D: t \in d\}|}$$

with N: total number of documents in the corpus $\mathrm{N} = |\mathrm{D}|$,
$|\{d \in D: t \in d\}|$: number of documents where the term *t* appears (i.e., $\mathrm{tf(t, d)} \neq 0$). If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust the denominator to $1 + |\{d \in D: t \in d\}|$.
Then tf–idf is calculated as

$$\mathrm{tfidf(t, d, D)} = \mathrm{tf(t, d)} \times \mathrm{idf(t, D)}$$

Variations of the tf–idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query. tf–idf can be successfully used for stop-words filtering in various subject fields, including text summarization and classification.
So, we think it is useful for getting keywords. For **age** we can use all answers as corpus and then use tf-idf to find the number in each answer.

c. Glove word embedding

For **highpower** and **attentioncorrect**, we must get the vector representation for the sentences.
GloVe (Global Vectors for Word Representation) is a tool recently released by Stanford NLP Group researchers Jeffrey Pennington, Richard Socher, and Chris Manning for learning continuous-space vector representations of words.
The GloVe model learns word vectors by examining word co-occurrences within a text corpus. Before we train the actual model, we need to construct a *co-occurrence matrix* ***X***, where a cell $\boldsymbol{X_{ij}}$ is a "strength" which represents how often the word *i* appears in the context of the word *j*. We run through our corpus just once to build the matrix ***X***, and from then on use this co-occurrence data in place of the actual corpus. We will construct our model based only on the values collected in ***X***.
Once we've prepared ***X***, our task is to decide vector values in continuous space for each word we observe in the corpus. We will produce vectors with a soft constraint that for each word pair of word *i* and word *j*,

$$\vec{w}_i^T \vec{w}_i + b_i + b_j = log X_{ij}$$

where b_i and b_j are scalar bias terms associated with words *i* and *j*, respectively. Intuitively speaking, we want to build word vectors that retain some useful information about how every pair of words *i* and *j* co-occur.

We'll do this by minimizing an objective function J, which evaluates the sum of all squared errors based on the above equation, weighted with a function f:

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(\vec{w}_i^T \vec{w}_j + b_i + b_j - \log X_{ij})^2$$

We choose an f that helps prevents common word pairs (i.e., those with large X_{ij} values) from skewing our objective too much :

$$f(X_{ij}) = \begin{cases} (\frac{X_{ij}}{x_{max}})^{\alpha} & if\ X_{ij} < x_{max} \\ 1 & otherwise. \end{cases}$$

When we encounter extremely common word pairs (where $X_{ij} > x_{max}$)this function will cut off its normal output and simply return 1. For all other word pairs, we return some weight in the range (0, 1), where the distribution of weights in this range is decided by α.

For **highpower**, we use all the answers as corpus as we do to the **age** column, so we can get the vectors for each word. Then we calculate the sentence vector by averaging the word vectors for words in each person's answer. Since the answers are various, we can't get the keyword if we don't know the semantic of the answers (it is easy to know because this feature is about telling a story which may vary between different people). So, we think averaging is a good method to get sentence vectors. The code is in *glove_word_embedding.py*, and we didn't use the built-in libraries.
For **attentioncorrect**, this feature can tell how serious a person takes the questionnaire, because if one is serious, he/she will input some sentence like 'I read the instructions. This feature is related to **pate_01** and **pate_02**. So we can use the word vector to check whether the sentence contains word 'instructions'. We use this method rather than check the word straightforward because there may be some misspelling in the answers such as 'instructions' which misses a ***t*** in the word. By using this method, we can find that them may have the similar vectors such that we can get good results in some ways.

4. K-Nearest Neighbors (KNN)

4.1 Model Description

For the task of interpolating missing features, K-Nearest Neighbors algorithm could be very effective since it can cope with different types of data, whether it's continuous, discrete or categorical. Also, as a non-parametric algorithm, it doesn't require learning, and can deal with addition or deletion in dataset with ease.

However, it's a little tricky to evaluate the performance of KNN in such a task, since there is no model to be saved for validation. Instead, the only way to run KNN is to run the algorithm on the entire dataset and perform prediction on the entire dataset. In a case where there is no missing data, the error rate for KNN is always 0, since every observation (or data point) can find a perfect match.

Assuming we have a set of known features *S* and a target feature *m* which does not belong to *S*, in order to have a prediction on *m*, we should at least have some of the features filled in *S*. So before using the algorithm, observations which are totally empty in S should be removed.

4.2 Algorithm Description

We assume that a feature of an observation can be approximated by the values of the observations that are closest to it, based on the values of other features. KNN can fill the missing feature by these simple steps:

1) Compute a distance matrix between the observations based on some distance metric;
2) For every missing value, find the K observations in the matrix that are closest to the observation it belongs to;
3) Fill the missing value from the neighbors with a mean/median/mode(most common).

Several metrics can be used for computing distance matrix based on the type of the data. In our case, the data is always categorical. Therefore, Hamming and Weighted Hamming Distance would be suitable for the task. For each instance, we look at the Hamming distance to compute distance matrix. To decide the missing value, we use the mode of the neighbors, which is reasonable for data taken from questionnaires because the mean and the median may not make sense for real answers. We also suggest using an odd K to avoid ties among neighbors.

4.3 Experiment and Analysis

We use two sets of features to demonstrate the performance of KNN for feature interpolation. The first set of features are the 12 features from "mcmost1" to "mcdv2" in the codebook. The second set of features are "attentioncorrect", "pate_01" and "pate_02". The reason we choose specific sets of features for the experiment is that KNN is unable to decide which features are relevant to the one we want to predict, so we need to choose beforehand sets of features that are relevant to each other.

1) The first set of features are mostly about the topic of "women at work", so there is a pretty good chance that some features are directly related to others. For instance, people who agree with the proposition in "mcmost5", which is "Most women are not really smart", are most likely biased. This means that they are likely to answer "Yes" to the question in "mcdv1", which is "Do you feel that this job is better suited for one gender rather than the other?".
2) The second set of features are about how the participants of the study are willing to take part in the study. With the help of natural language processing method introduced before, we can turn feature "attentioncorrect" into a two-class feature: whether the participant wrote down the expected answer. It is rational to assume that people with low level of attention to the questionnaire would ignore the instructions in "attentioncorrect", therefore fail to write down the expected answer, which is "I read the instructions".

We then try to fit the KNN algorithm to the data. Unlike parametric machine learning algorithms, we cannot split the data into two sets, train on one of them and validate the performance on the other. There is no point training on known data anyway since every data point will be matched to itself. One way to analyze the performance is to look at the predictions case by case:

```
attributes:
 mcmost1    NaN
mcmost2    NaN
mcmost3    NaN
mcmost4    NaN
mcmost5    NaN
mcsome1    1.0
mcsome2    2.0
mcsome3    1.0
mcsome4    1.0
mcsome5    1.0
mcdv2      NaN
Name: 1009, dtype: float64
prediction: 0.0
==================
```

In this case, we use the first set of features, and try to predict "mcdv1" from other 11 features. The participant didn't answer the questions with the quantifier "most", but according to his/her response to "mcsome3", he/she agree that "The best job for some women is something like cook, nurse or teacher". This could rationally lead to him/her response to "Do you feel that this job is better suited for one gender rather than the other?", which is "Yes".

In another case, we use the second set of features. We try to predict "pate_01" from the other two features. Based on the fact that the participant wrote the expected answer for "attentioncorrect" and that his/her response to "pate_02" is "a lot of attention", we can safely presume that he/she paid attention to the instructions, which means "a lot of efforts" for "pate_01" is reasonable.

```
==================
attributes:
 pate_02             4.0
attentioncorrect    1.0
Name: 1607, dtype: float64
prediction: 4.0
==================
```

4.4 Model Evaluation

In this section we will discuss the pros and cons of KNN for the task of predicting missing features. First of all, the choice of KNN is based on the assumption that people who give similar answer to a certain set of questions are very likely to give similar answer to another set of correlated questions. In the ML3 dataset, the responses given by the participants are not completely random, and some questions are very much related with some other questions. If we choose the right set of questions that are related to each other, we can say given enough number of samples, KNN can make a good prediction on some features. This leads to the biggest advantage of KNN, which is that it can work in a very simple manner, and it does work. It cannot model the distribution of the entire dataset, but it can operate locally, which is enough for predicting a single value.

There are also disadvantages for KNN. For instance, it is easily affected by noise in the data. This local effect can be blurred by having a high K. Also, the algorithm is a lazy-learning algorithm which simply remembers the dataset, which cannot generalize to other data.