

CS 536 : Final Project - Data Completion and Interpolation

Haoyang Zhang, Han Wu, Shengjie Li

May 11, 2019

1 Introduction, group members and division of workload

In this group project, we implemented an autocoder for interpolating missing features from features we have and achieved .

Name NetID	Workload
Han Wu hw436	Fine-tuned the parameters of our model. Did some experiments for the evaluation of our model. Wrote part of the report.
Haoyang Zhang hz333	Analyzed and wrote scripts to clean the data. Wrote scripts to restore human-friendly data from the output of our model. Wrote part of the report.
Shengjie Li sl1560	Implemented the basics of neural networks including back-propagation and several loss functions and activation functions. Wrote part of the report.

2 Data preprocessing and thoughts towards this project

1. How to represent or process the data. Data features may contain a number of diverse data types (real values, integer values, categorical or binary values, ordered categorical values, open/natural language responses). How can you represent these for processing and prediction?

- **Data encoding**

Given the `ML3AllSites` dataset, we can classify each column into 6 types: (positive) integers, unordered multiple choices, Boolean-like values, long texts, other valid responses and NA's.

(Note: the 1177th row is wrongly encoded in the original dataset, but we fixed it in the `ML3AllSitesC`. However, it again includes some more bugs in date related columns like column 124 Date.x because of Microsoft Excel. Anyway, we eventually fixed them when reading the dataset when calling `dataFormat.py`)

A detailed codebook is available in Appendix A, and further reference is in `/code/dataFormat.py`, which is organized in the order of original dataset.

- (Postive) Integers

Many columns belong to this type, for examples, best grade 1 (column 21), mcdv1 (column 71), temperature in lab (column 126) and intrinsic (column 247). If the choices are exactly ordered, say “1” is “unhappy”, and “10” is “happy”, we consider this column as this type. Tough “2” may not be twice happier than “1”, nevertheless, 2 is indeed happier. Note that some columns may include float numbers, say intrinsic, but we can multiply a factor to scale all responses to integers. Also, some

responses could be negative values, say `mcdv1`, but we still can add a number to shift all of them to non-negative values.

Each value in these columns is the real value instead of probability.

- Unordered Multiple Choices

Many columns belong to this type, for examples, ethnicity (column 42), gender (column 44), major (column 70) and V position (column 115). Note that these columns may include natural language response, say major, but we have classified all responses into several choices.

Each choice of these columns are exclusive and unordered. Therefore, we cannot just simply encode them into integers. Or we will have to face the explanatory problem: If we encode “computer science” into “1” and “mathematics” into “2”, do we mean a “mathematics” is equal to 2 “computer science”? Therefore, we choose one-hot encoding to use the same number as choices of Boolean values to represent the participant’s choice. In this case, we can consider each value as the probability that this participant will choose this response.

- Boolean-like Values

A few columns belong to this type. Some are natural language responses but there is a true answer, say anagrams (column 5 and 6) and attention correct (column 10), and the test is highly concerning about whether the participant correct or not instead of what they answered. Others are multiple choices with exactly 2 possible answers like `mcmost` (column 76 to 80), and for simplicity we prefer to use 1 Boolean value to represent his/her choice.

In general, we can consider this type as a special multiple choice type. Namely, each value in this column is a probability.

- Long Texts

There are exactly 3 columns belongs to this type: `highpower` (column 45), `lowpower` (column 67) and `Notes` (column 134). Because of time limitation, we skipped to process these 3 columns.

- Other Valid Responses

Some natural language responses that describe a real number belong to this type, for example, K ratio (column 66), worst grade 2 (column 118) and SR TF Correct (column 133). Some obviously unrelated or redundant data is also this type. For an example, Date Computer (column 220) is a duplicate to Month Computer, Day Computer and Year Computer (column 222 to 224).

- NA’s

In order to distinguish normal data and NA’s, we use a valid mask. For each encoded feature, we use a Boolean value to indicate whether it is normal or NA. Namely, each row in original dataset is coded into 2 rows, where one is a valid mask and the other is the real data.

For simplicity, we set all NA’s to 0 just like dropout. When computing error, we use the mask to set these features’ loss to 0.

- **Data preprocessing**

- Scaling to $[0, 1]$

Notice that the coded data can be further divided into 2 types: real values and probabilities. Notice that real values could be from negative infinity to positive infinity. (There may be some more restrictions like temperature in lab cannot be lower than -460. But in general its range is way larger than $[0, 1]$.) But for our prediction simplicity, we will linearly scale the largest value seen to 1 and the smallest to 0.

Here are some more things we can do, but because of time limit, we skipped them. The straightforward problem for this naïve scaling is we might be trapped by

outliers. For example, (this example is already fixed.) some participant claimed his/her/its age is about 150. If we directly apply the scaling, most 20-ish responses will be scaled into about 0.007, and the only response that is greater than 0.3 is that 150, which will make this feature hard to predict precisely. Therefore, we should throw out these outliers.

But a further thought is that this situation can also happen when the response distributed unevenly. For example, many people answer either about 1 to 2 or 8 to 9. In this case, we use a lot of space to encode unlikely values, which leads to the same result. In this case, a nonlinear scale method will be helpful. A rudimentary thought is we sort all values in the dataset and linearly scale first 10% values to the range $[0, 0.1)$, second 10% to $[0.1, 0.2)$, and so on.

Another problem is that we cannot predict any larger or smaller values than values in the dataset. A plausible justification could be that it is generally unlikely to see an extreme small or large value. But if we adopt the nonlinear scale method, we can map negative infinity (or the smallest valid value) to the smallest value in dataset into $[0, 0.1)$, and all values in dataset to $[0.1, 0.9)$, and so on.

- Grouping Features

Notice that this dataset is all about 10 psychological tests. Therefore, we can assume the features in the same tests are more related than features between different tests. Namely the dimensionality in each tests is relatively small. Therefore, we can group features in terms of tests. Some global information about this participant, like demographic features and personality features, is grouped into another global set instead of test sets, which is called group 0. Some definitely unrelated data like participant ID (column 1) is grouped into another set, called group 11.

In this way, we can try to use group 0 and each test group to predict blank features in this test just by picking 2 group indices instead of a huge number of feature indices.

- Selecting Features

Notice that all group 11 features can be discard based on our prior knowledge. (Actually we should use graphical model to prove it.) A further thing we should do is run Chow Liu Algorithm on group 0 and each test group to find weak-dependent intra-test features (Namely, all its edges are weak.) and eliminate them, and then run it on the whole feature space to try to further eliminate features.

- Discarding Almost NA Data Points

Notice that there are several almost blank rows in the original datasets. These data points cannot tell us many things. We can use NA masks to identify them. More specifically, we remove these rows whose mask has more than 100 `False`.

2. How to model the problem of interpolation. What are the inputs, what are the outputs? An important if subtle question to consider here - what does it mean to predictor or interpolate a missing feature?

Given the preprocessed data, each data point is a vector with some blanks, and the whole dataset is a matrix with blanks. Our goal is to fill the blanks. Notice that we can do this because the dimensionality of this matrix is limited. Namely, many features are related to each other. For example, those who claimed they are high self-esteemed (column 252) are generally less stressed (column 253) and their mood (column 248) is better. Therefore, we can consider the dataset as a limited-rank matrix with some blanks whose size is 2434×261 .

A straight forward idea to this problem is consider the blanks as noise, and our goal becomes to detect and eliminate this noise. Notice that the rank of this matrix is limited, therefore we can transform it into a much smaller matrix and restore it. Assuming the noise is relatively smaller than the information that this matrix gives, when we are transform or compressing

this matrix, the noise will be eliminated, and then we can decompress it to restore the blanked values.

3. Model selection. What kind of model or models do you want to consider?

Before we started doing this project, we discussed several models.

- Autoencoders

It can compress and decompress the given inputs into smaller representations, and these representations contain key information of the inputs. Thus, we can make use of these representations to classify data and generate data, which makes autoencoders handy in this situation.

- A transfer learning approach:
- Autoencoders with RNN:

4. Quantifying loss or error. How can you quantify how good a model is, how to measure its loss/error? This is important not only in terms of evaluating your model, but in terms of training as well - how can you refine and improve your model without a way of comparing them?

Because we are using an autoencoder, we are measuring the reconstruction errors. For the reason that there are many data types, we are treating the data over different loss functions. For real values and integer values, we are using mean squared error $L_{MSE}(\theta) = \frac{1}{m} \sum_{i=1}^m (\underline{y}^i - \underline{p}^i)^2$ (m denotes the number of data points) and root-mean-squared error $L_{RMSE}(\theta) = \sqrt{L_{MSE}(\theta)}$. For categorical values, we first converted them to one-hot encoding, then we are treating the problem as a multi-label classification problem (for these categorical values only). Thus, for categorical values, we are using binary cross-entropy loss $L_{MSE}(\theta) = \frac{1}{m} \sum_{i=1}^m [\underline{y}^i \log(\underline{p}^i) + (1 - \underline{y}^i) \log(1 - \underline{p}^i)]$. As the data set is a combination of different types of data, we are using a combination of different loss functions.

5. Training. What kind of training algorithm can you apply to your model(s)? What design choices do you have to make here?

We

6. Feature selection. It is frequently useful in learning problems to focus on specific features and exclude others, to try to eliminate spurious features and focus on what matters. How can that be applied here?
7. Validation. How can you prevent or avoid over-fitting? Can you apply the usual training/testing/validation paradigm to this problem? How do you choose the training or testing data? Note that a record won't need to be complete to still be useful, potentially, in interpolation. Can cross-validation be applied here? This can be especially important when the data set is not overwhelmingly large and data must be used carefully.
8. Evaluation. How good is your final model? How can you evaluate this? What are the limits and strengths of your model - how many features does a new record actually need to be able to interpolate well?

3 Requirements

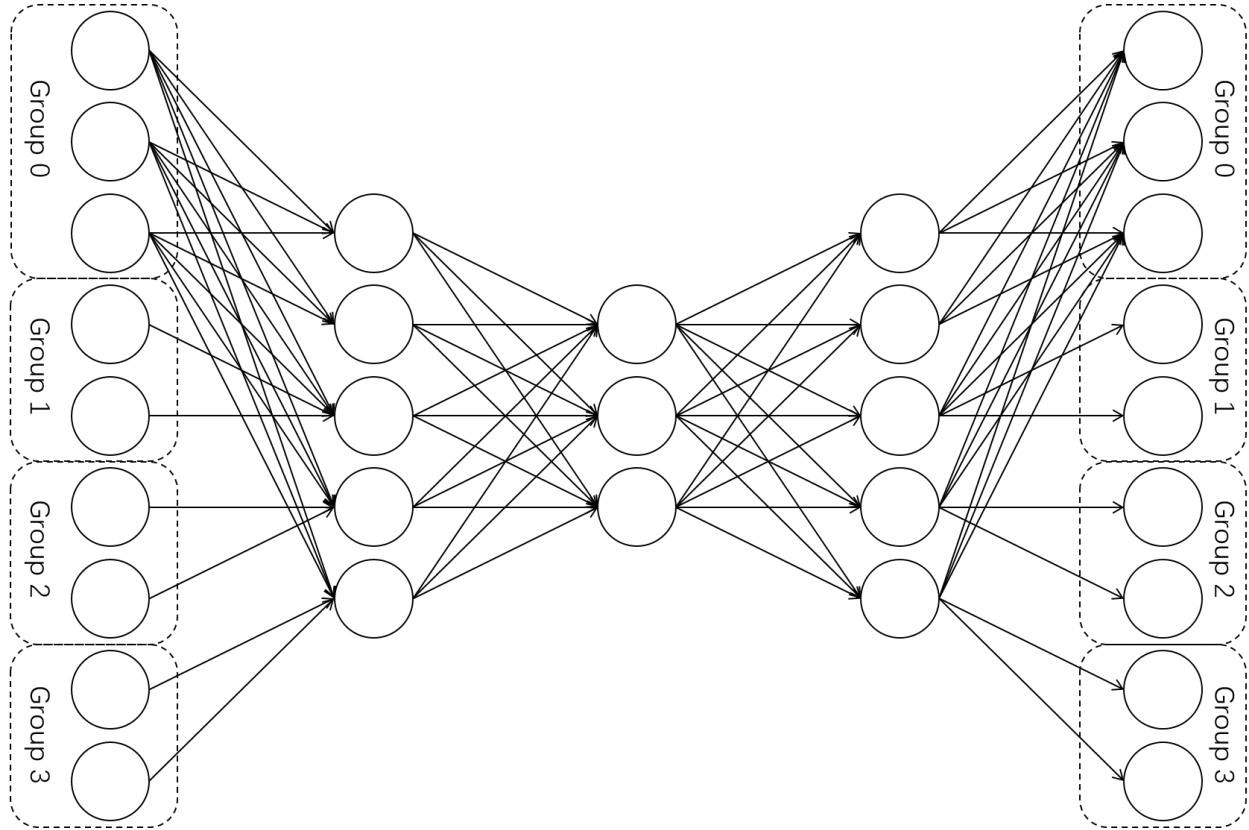


Figure 1: Our autoencoder

1. **Describe your Model:** What approach did you take? What design choices did you make, and why? How did you represent the data? How can you evaluate your model for goodness of fit? Did you make an effort to identify and exclude irrelevant variables? How did you handle missing data?

Given the Formalization of interpolation, We chose to use an autoencoder as our model because it can compress and decompress the given inputs. Figure 1 shows the structure of our model.

- (a) The input
 - (b) The output
 - (c) Some observation
- 2.
 - 3.
 - 4.
 - 5.
 - 6.

4 Bonus

Appendix A Codebook

Codebook

This part describes how we encode the original datasets. Each row is `column {original column} ({coded start column}: {coded end column}): data{column name}: {original column dtype} / {preCoded dtype} / {coded dtype}`. `original column` is the column in original datasets. `coded column` is the column in coded dataset that inputs our model, and it includes `start column` but excludes `end column`. `column name` is the name in original dataset (according to its first row). `original column dtype` is the data type in original dataset. `preCoded dtype` is the middle state data type. `coded dtype` is the coded data type before scaling.

Some column that we discarded is set to `None`. Note that column 21 and 117 is further coded into `int` when we are scaling them.

```
column 0 (223: 244): dataSite: str / int / [bool's]
column 1 (244: 245): dataParticipant_ID: str / int / int
column 2 (245: 246): dataRowNumber: str / int / int
column 3 (246: 247): datasession_id: str / int / int
column 4 (0: 1): dataage: str / int / int
column 5 (150: 151): dataanagrams1: str / int / bool
column 6 (151: 152): dataanagrams2: str / int / bool
column 7 (152: 153): dataanagrams3: str / int / bool
column 8 (153: 154): dataanagrams4: str / int / bool
column 9 (1: 7): dataattention: str / int / [bool's]
column 10 (130: 131): dataattentioncorrect: str / int / bool
column 11 (164: 165): databackcount1: str / int / bool
column 12 (165: 166): databackcount10: str / int / bool
column 13 (166: 167): databackcount2: str / int / bool
column 14 (167: 168): databackcount3: str / int / bool
column 15 (168: 169): databackcount4: str / int / bool
column 16 (169: 170): databackcount5: str / int / bool
column 17 (170: 171): databackcount6: str / int / bool
column 18 (171: 172): databackcount7: str / int / bool
column 19 (172: 173): databackcount8: str / int / bool
column 20 (173: 174): databackcount9: str / int / bool
column 21 (194: 195): databestgrade1: str / (int, int) / (int, int)
column 22 (195: 196): databestgrade2: str / int / int
column 23 (196: 197): databestgrade3: str / int / int
```

```

column 24 (197: 198): databestgrade4: str / int / int
column 25 (198: 199): databestgrade5: str / int / int
column 26 (154: 155): databig5_01: str / int / int
column 27 (155: 156): databig5_02: str / int / int
column 28 (156: 157): databig5_03: str / int / int
column 29 (157: 158): databig5_04: str / int / int
column 30 (158: 159): databig5_05: str / int / int
column 31 (159: 160): databig5_06: str / int / int
column 32 (160: 161): databig5_07: str / int / int
column 33 (161: 162): databig5_08: str / int / int
column 34 (162: 163): databig5_09: str / int / int
column 35 (163: 164): databig5_10: str / int / int
column 36 (7: 8): datadiv3filler: str / int / bool
column 37 (189: 190): dataelm_01: str / int / int
column 38 (190: 191): dataelm_02: str / int / int
column 39 (191: 192): dataelm_03: str / int / int
column 40 (192: 193): dataelm_04: str / int / int
column 41 (193: 194): dataelm_05: str / int / int
column 42 (8: 17): dataethnicity: str / int / [bool's]
column 43 (247: 248): datafeedback: str / int / bool
column 44 (17: 22): datagender: str / int / [bool's]
column 45 (248: 249): datahighpower: str / int / bool
column 46 (249: 250): datainstructbig5: str / int / bool
column 47 (250: 251): datainstructintrinsic: str / int / bool
column 48 (251: 252): datainstructmli: str / int / bool
column 49 (252: 253): datainstructnfc: str / int / bool
column 50 (22: 23): dataintrinsic_01: str / int / int
column 51 (23: 24): dataintrinsic_02: str / int / int
column 52 (24: 25): dataintrinsic_03: str / int / int
column 53 (25: 26): dataintrinsic_04: str / int / int
column 54 (26: 27): dataintrinsic_05: str / int / int
column 55 (27: 28): dataintrinsic_06: str / int / int
column 56 (28: 29): dataintrinsic_07: str / int / int
column 57 (29: 30): dataintrinsic_08: str / int / int
column 58 (30: 31): dataintrinsic_09: str / int / int
column 59 (31: 32): dataintrinsic_10: str / int / int
column 60 (32: 33): dataintrinsic_11: str / int / int

```



```

column 61 (33: 34): dataintrinsic_12: str / int / int
column 62 (34: 35): dataintrinsic_13: str / int / int
column 63 (35: 36): dataintrinsic_14: str / int / int
column 64 (36: 37): dataintrinsic_15: str / int / int
column 65 (131: 134): datakposition: str / int / [bool's]
column 66 (134: 135): datakratio: str / int / int
column 67 (253: 254): datalowpower: str / int / bool
column 68 (135: 138): datalposition: str / int / [bool's]
column 69 (138: 139): datalratio: str / int / int
column 70 (37: 46): datamajor: str / int / [bool's]
column 71 (205: 206): datamcdv1: str / int / int
column 72 (206: 207): datamcdv2: str / int / int
column 73 (207: 208): datamcfiller1: str / int / int
column 74 (208: 209): datamcfiller2: str / int / int
column 75 (209: 210): datamcfiller3: str / int / int
column 76 (210: 211): datamcmost1: str / int / bool
column 77 (211: 212): datamcmost2: str / int / bool
column 78 (212: 213): datamcmost3: str / int / bool
column 79 (213: 214): datamcmost4: str / int / bool
column 80 (214: 215): datamcmost5: str / int / bool
column 81 (215: 216): datamcsome1: str / int / bool
column 82 (216: 217): datamcsome2: str / int / bool
column 83 (217: 218): datamcsome3: str / int / bool
column 84 (218: 219): datamcsome4: str / int / bool
column 85 (219: 220): datamcsome5: str / int / bool
column 86 (46: 54): datamood_01: str / int / [bool's]
column 87 (54: 62): datamood_02: str / int / [bool's]
column 88 (62: 63): datanfc_01: str / int / int
column 89 (63: 64): datanfc_02: str / int / int
column 90 (64: 65): datanfc_03: str / int / int
column 91 (65: 66): datanfc_04: str / int / int
column 92 (66: 67): datanfc_05: str / int / int
column 93 (67: 68): datanfc_06: str / int / int
column 94 (139: 142): datanposition: str / int / [bool's]
column 95 (68: 69): datanratio: str / int / int
column 96 (69: 70): datapate_01: str / int / int
column 97 (70: 71): datapate_02: str / int / int

```

```

column 98 (71: 75): datapate_03: str / int / [bool's]
column 99 (75: 82): datapate_04: str / int / [bool's]
column 100 (82: 89): datapate_05: str / int / [bool's]
column 101 (142: 145): datarposition: str / int / [bool's]
column 102 (145: 146): datarratio: str / int / int
column 103 (174: 175): datasarcasm: str / int / int
column 104 (199: 200): dataselfesteem_01: str / int / int
column 105 (89: 90): datastress_01: str / int / int
column 106 (90: 91): datastress_02: str / int / int
column 107 (91: 92): datastress_03: str / int / int
column 108 (92: 93): datastress_04: str / int / int
column 109 (182: 183): datatempest1: str / int / int
column 110 (183: 184): datatempest2: str / int / int
column 111 (184: 185): datatempest3: str / int / int
column 112 (185: 186): datatempfollowup1: str / int / int
column 113 (186: 187): datatempfollowup2: str / int / int
column 114 (187: 188): datatempfollowup3: str / int / int
column 115 (146: 149): datavposition: str / int / [bool's]
column 116 (149: 150): datavratio: str / int / int
column 117 (200: 201): dataworstgradel: str / (int, int) / (int, int)
column 118 (201: 202): dataworstgrade2: str / int / int
column 119 (202: 203): dataworstgrade3: str / int / int
column 120 (203: 204): dataworstgrade4: str / int / int
column 121 (204: 205): dataworstgrade5: str / int / int
column 122 (93: 100): datayear: str / int / [bool's]
column 123 (254: 255): dataStation: str / int / bool
column 124 (255: 256): dataDate_x: str / int / bool
column 125 (256: 257): dataExperimenter: str / int / bool
column 126 (188: 189): dataTemperatureinlab: str / int / int
column 127 (100: 103): dataOrderofTasks: str / int / [bool's]
column 128 (175: 178): dataClipboardWeight: str / int / [bool's]
column 129 (178: 179): dataIIResponse: str / int / int
column 130 (120: 124): dataSRCondition: str / int / [bool's]
column 131 (124: 128): dataSRMeetingResponse: str / int / [bool's]
column 132 (128: 129): dataSRConfidenceResponse: str / int / int
column 133 (129: 130): dataSRTFCorrect: str / int / int
column 134 (257: 258): dataNotes: str / int / bool

```

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

None

```
column 170 (179: 182): dataClipboardMaterial: str / int / bool
```

```
column 171 (103: 104): dataPersistence: str / int / int
```



```

column 246 (108: 109): dataNeuroticism: str / int / int
column 247 (109: 110): dataIntrinsic: str / int / int
column 248 (110: 111): dataMood: str / int / int
column 249 (111: 112): dataNFC: str / int / int
column 250 (112: 113): dataReportedAttention: str / int / int
column 251 (113: 114): dataReportedEffort: str / int / int
column 252 (114: 115): dataSelfEsteem: str / int / int
column 253 (115: 116): dataStress: str / int / int
None
column 255 (220: 221): dataMostEndorse: str / int / int
column 256 (221: 222): dataSomeEndorse: str / int / int
column 257 (222: 223): dataCredCond: str / int / bool
column 258 (116: 117): dataGenderfactor: str / int / bool
None
None
None
None
None
None
None
None
None
None
column 268 (117: 118): dataTempCond: str / int / bool
column 269 (118: 119): dataTargetGender: str / int / bool
column 270 (119: 120): dataArgumentQuality: str / int / int
None
None
None

```