Data X

Ashis Ghosh
April 24th 2018

HW04 - Decision Trees, Entropy, NLP, SQL

Part 1

Part 1.1

FINAL RESULTS:

Entropy Values:
Above 30: 1.0 HasJob: 0.951205059305 HasFam: 0.811278124459

Information Gained:
Above 30: 0.0 HasJob: 0.0487949406954 HasFam: 0.188721875541

CALCULATIONS:

Weights of each X feature

w_above30 = 0.5
w_hasJob = 0.625
w_hasFam = 0.75

Probabilities of defaulting given each X feature is positive or negative.

p_above30 = P(default|above30) = 0.5
p_above30 = P(default|!above30) = 0.5

p_hasJob = P(default|hasJob) = 0.4
p_hasJob = P(default|!hasJob) = 0.67

p_hasFam = P(default|hasFam) = 0.25
p_hasFam = P(default|!hasFam) = 0.75

Entropy Calculations

$$H(x) = \sum_{x} p(x)\log\left(\frac{1}{p(x)}\right)$$

p_above30_ent = (p_above30*np.log2(1/p_above30) + (1-p_above30)*np.log2(1/(1-p_above30)))* w_above30 + (p_notAbove30*np.log2(1/p_notAbove30) + (1-p_notAbove30)*np.log2(1/(1-p_notAbove30)))*(1-w_above30)

p_above30_ent = 1.0

p_hasJob_ent = (p_hasJob*np.log2(1/p_hasJob) + (1-p_hasJob)*np.log2(1/(1-p_hasJob)))* w_hasJob + (p_notHasJob*np.log2(1/p_notHasJob) + (1-p_notHasJob)*np.log2(1/(1-p_notHasJob)))*(1-w_hasJob)

p_hasJob_ent = 0.951205059305

p_hasFam_ent = (p_hasFam*np.log2(1/p_hasFam) + (1-p_hasFam)*np.log2(1/(1-p_hasFam)))* w_hasFam + (p_notHasFam*np.log2(1/p_notHasFam) + (1-p_notHasFam)*np.log2(1/(1-p_notHasFam)))*(1-w_hasFam)

p_hasFam_ent = 0.811278124459

Part 1.2

A = 0.7
B = 0.2
C = 0.1

S_ent = A*np.log2(1/A) + B*np.log2(1/B) + C*np.log2(1/C) = 1.1568

S has an entropy of 1.1568, meaning it should not be compressed into any number of bits less than that as it will be at risk of losing information.

Part 2

Part 2.a - Preprocessing & Modelling

For preprocessing the following steps were used:

1. The body of text was read as a csv using pandas into a dataframe with a sentence per row
2. Manually tokenized:

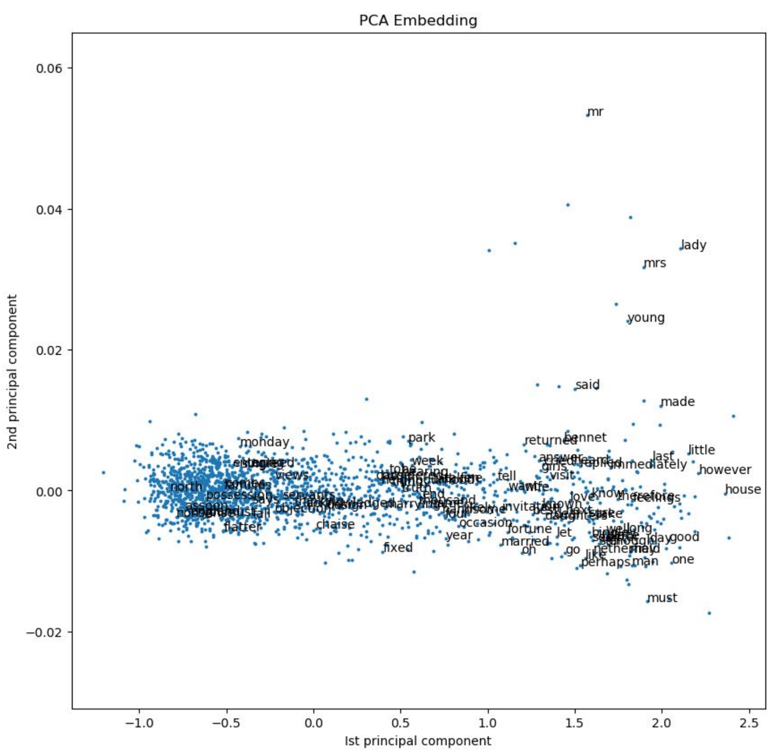
    a. Special characters were removed and replaced with spaces
    b. The text was changed to lower case
    c. Each sentence was split into an array
3. Stopwords
    a. Stopwords were removed using the list from NLTK
        i. The list was copied into a txt file and imported as to avoid downloading the corpus

For modelling the follow parameters were used:

Part 2.b - Report the vocabulary count, embedding size, number of training iterations in your modelling

Vocabulary Count = 1819 words
Embedding Size = 100
Epochs (number of iterations) = 8

Part 2.c - Visualizations



Firstly, the 2nd principal component does not show strong correlation with the words as it's range is only approx ~0.08 as opposed to the 1st's ~4.0 range.

We can also see how close certain words are to each other, there is a clear cluster around (-0.5, 0). However, we can also see that there are certain words that are further away from the cluster and are quite topical to the text - i.e. “young”, “mrs”, and “lady” in the upper right quadrant. I believe this can be considered as another cluster and these are topically related. However, based on the PCA, the distance between these words are relatively far. This is not a great measure for *how* correlated words are.

Part 2.d - 5 Intrinsic Evaluations

Model Similarity - “elizabeth” & “girl”: 0.994126302661
Most Similar - “girl”: ('even', 0.9997328519821167)
Doesn’t Match - 'story', 'great', 'spirit', 'disposition', 'delighted', 'altogether': spirit
Most Similar - “woman”: ('great', 0.9997955560684204)
Model Similarity - “great”, “spirit”: 0.996579198264

Part 3

3.1.1. SELECT all records in the table.
'SELECT * FROM parents'

	parent	child
0	abraham	barack
1	abraham	clinton
2	delano	herbert
3	eisenhower	fillmore
4	fillmore	abraham
5	fillmore	delano
6	fillmore	grover

3.1.2. SELECT child and parent, where abraham is the parent.
'SELECT * FROM parents WHERE parent="abraham"'

	parent	child
0	abraham	barack
1	abraham	clinton

3.1.3. SELECT all children that have an 'e' in their name (hint: use LIKE and '%e%').
'SELECT * FROM parents WHERE child LIKE "%e%"'

	parent	child

0	delano	herbert
1	eisenhower	fillmore
2	fillmore	delano
3	fillmore	grover

3.1.4. SELECT all unique parents (use SELECT DISTINCT) and order them by name, descending order (i.e. fillmore first)

'SELECT DISTINCT parent FROM parents ORDER BY parent DESC'

	parent
0	fillmore
1	eisenhower
2	delano
3	abraham

3.1.5. SELECT all dogs that are siblings (one-to-one relations). Only show a sibling pair once. To do this you need to select two times from the parents table

```
"""
SELECT parent, COUNT(*)
FROM parents
GROUP BY parent
HAVING COUNT(*) > 1
"""
```

```
"""
SELECT child
FROM parents
WHERE parent="abraham"
OR
parent = "fillmore"
"""
```

	child
0	barack
1	clinton
2	abraham

3	delano
4	grover

3.2.1. COUNT the number of short haired dogs

'SELECT *, COUNT(fur) FROM dogs WHERE fur="short"'

	name	fur	COUNT(fur)
0	grover	short	3

3.2.2. JOIN tables parents and dogs and SELECT the parents of curly dogs.

```
"""
SELECT parent
FROM dogs
INNER JOIN parents ON dogs.name = parents.child
WHERE fur="curly";
"""
```

	parent
0	eisenhower
1	delano

3.2.3. JOIN tables parents and dogs, and SELECT the parents and children that have the same fur type. Only show them once.

```
"""
SELECT parent, child
FROM parents
LEFT JOIN dogs ON parents.parent == dogs.name
LEFT JOIN dogs AS dogs2 ON parents.child == dogs2.name
WHERE dogs.fur==dogs2.fur
"""
```

	parent	child
0	abraham	clinton

3.3.1. SELECT the animal with the minimum weight. Display kind and min_weight.

```
"""
SELECT *, MIN(weight) FROM animals
"""
```

	kind	legs	weight	MIN(weight)
0	parrot	2	6	6

3.3.2. Use the aggregate function AVG to display a table with the average number of legs and the average weight.

```
"""
SELECT AVG(legs), AVG(weight) FROM animals
"""
```

	AVG(legs)	AVG(weight)
0	3.0	2009.333333

3.3.3. SELECT the animal kind(s) that have more than two legs, but weighs less than 20. Display kind, weight, legs.

```
"""
SELECT * FROM animals
WHERE legs > 2 AND weight < 20
"""
```

	kind	legs	weight
0	cat	4	10
1	ferret	4	10

3.3.4. SELECT the average weight for all the animals with 2 legs and the animals with 4 legs (by using GROUP BY).

```
"""
SELECT AVG(weight), legs FROM animals
GROUP BY legs
"""
```

	AVG(weight)	legs
0	4005.333333	2
1	13.333333	4