Pset_2_RT_and_surprisal

April 30, 2023

[1]:
```python
import seaborn as sns; sns.set(style="white", color_codes=True)
import matplotlib.pyplot as plt
import statsmodels.api as sm
import pandas as pd
import numpy as np
import string

pd.options.mode.chained_assignment = None
```

1 Loading the Datasets

1.0.1 Load ngram surprisals

Let's fetch the `ngram` surprisal file:

[2]:
```python
surprisals = pd.read_csv('https://gist.githubusercontent.com/scaperex/
 ↪f19f77f5157f7ba7ea1adf72a72847da/raw/
 ↪d5d553b1217ea70fe3261ce5d9a0532f29769817/5gram_surprisals.tsv',␣
 ↪index_col=False, sep='\t')
surprisals
```

[2]:
```
      sentence_id  token_id   token  surprisal
0               1         1      In    4.57937
1               1         2   <unk>    7.45049
2               1         3  County   12.65410
3               1         4   <unk>    6.11317
4               1         5    near   12.22380
…             …         …     …         …
7693          464        17       a    3.23962
7694          464        18  leader   12.81650
7695          464        19     and    5.90348
7696          464        20   <unk>    4.62292
7697          464        21    </s>   11.10650

[7698 rows x 4 columns]
```

1.0.2 Load RT data

Let's fetch also the Brown_RTs dataset and see how it looks like

[3]:
```python
sprt = pd.read_csv('https://gist.githubusercontent.com/scaperex/01b55eab89b81dc882055e0d27d61016/raw/046dbb7f0586b5dc1a368ee882f2cb923caad3df/brown-spr-data-for-pset.csv', index_col=0).sort_values(by='code')
sprt.reset_index(drop=True, inplace=True)
sprt
```

[3]:
```
             word   code subject  text_id  text_pos  word_in_exp    time
0              In  17000    s001        0         0         2285  399.90
1              In  17000    s028        0         0         2503  290.32
2              In  17000    s014        0         0         1394  501.59
3              In  17000    s021        0         0         2525  210.93
4              In  17000    s010        0         0          579  862.35
…             …     …       …       …        …          …       …
136902   captain.  35763    s021       12       763         1430  425.18
136903   captain.  35763    s030       12       763         1489  383.32
136904   captain.  35763    s007       12       763         3426  506.40
136905   captain.  35763    s004       12       763         3528  669.29
136906   captain.  35763    s022       12       763          763  304.40

[136907 rows x 7 columns]
```

2 Harmonize N-gram surprisal and RT data

We have the model-derived surprisal values. To align it with human reading times, complete the following cell. This will create for us a data frame containing both metrics in sync.

In `surprisals` each row represents a word. In `sprt` each row represents a word that was displayed in a trial. Therefore, in `sprt` there are multiple row for each word - one for each subject.

Note that the words are ordered the same in both files (i.e. they both start with 'In', then 'Ireland's'/'<unk>, then 'County', and so on. However, there are differences, such as a special token for end of sentence which appears only in `surprisals`, among others.

See the PDF instructions for more details.

2.1 Pre-processing

2.1.1 Excluding Outliers

First, we will exclude "outlier" reading times (from 'sprt' dataset) that might reflect non-linguistic processing effects. We will follow the outlier-removal policy used in Smith and Levy (2013). According to the policy, we will remove records for which at least one of the following conditions hold:

- RT is less than 80 ms.

- RT is greater than 1500 ms.
- RT is not within 4-standard-diviations range from the mean RT of the participant.

[4]:
```
# removing records with time less than 80 or greater than 1500
sprt = sprt[sprt["time"].between(80, 1500)]
sprt.reset_index(drop=True, inplace=True)
```

[5]:
```
# create "subject_interval_dict" where:
# key - subject id in the experimant
# value - 4-standard-diviations range from the mean interval

compact_sprt = sprt[["subject", "time"]]

subject_lst = compact_sprt.groupby("subject").mean().index.tolist()
mean_lst = compact_sprt.groupby("subject").mean().values.tolist()
std_lst = compact_sprt.groupby("subject").std().values.tolist()

subject_interval_dict = {}

for subject, mean, std in zip(subject_lst, mean_lst, std_lst):
  std_4_interval = (mean[0] - 4*std[0], mean[0] + 4*std[0])
  subject_interval_dict[subject] = std_4_interval
```

[6]:
```
# removing all records that outside the 4-std interval
remove_index_lst = []

for i, row in sprt.iterrows():
  l, u = subject_interval_dict[row["subject"]]

  if row["time"] < l or row["time"] > u:
    remove_index_lst.append(i)

sprt.drop(remove_index_lst, inplace=True)
sprt.reset_index(drop=True, inplace=True)
```

2.1.2 Fix Synchronized Order

Now, we would like to process the data in a way that will allow us to ailgn each word with its surprisal and reading time.

The 'sprt' dataset consists of blocks of records, where the records in each block describe the same word (same text_id and text_pos) from different subjects.

Each word (token) in 'surprisals' corresponds to a block in 'sprt' in synchronized order.

We would like to use this order to align each word surprisal with its average reading time in the block (across the different subjects).

However, after exploring the data, we found a few errors that mess with the synchronized order. Since these errors are very different from each other, we will handle them manually and show the process.

[7]:
```
# find the errors that mess with the synchronized order
def find_mismatch(surprisals, sprt):

  sprt_idx, surprisals_idx = 0, 0
  sprt_len = len(sprt)

  for _, surprisals_row in surprisals.iterrows():

    while sprt_idx < sprt_len - 1 and \
    sprt.loc[sprt_idx]["text_id"] == sprt.loc[sprt_idx+1]["text_id"] and \
    sprt.loc[sprt_idx]["text_pos"] == sprt.loc[sprt_idx+1]["text_pos"]:

      if '<unk>' not in surprisals_row["token"] and surprisals_row["token"] !=␣
↪sprt.loc[sprt_idx]["word"]:
        print("Error in:")
        print(f"surprisals_idx: {surprisals_idx}, sprt_idx: {sprt_idx}")
        print(f'surprisals word: {surprisals_row["token"]} VS sprt word: {sprt.
↪loc[sprt_idx]["word"]}')

        return

      sprt_idx += 1

    sprt_idx += 1
    surprisals_idx += 1

  print("No errors were found!")
```

[8]:
```
find_mismatch(surprisals, sprt)
```

```
Error in:
surprisals_idx: 24, sprt_idx: 399
surprisals word: </s> VS sprt word: Undoubtedly
```

As we can see, the token “</s>” is only used in the ‘surprisals’ dataset (to indiacte end of sentence for the language model). Therefore we will remove it from the ‘surprisals’ dataset.

[9]:
```
surprisals = surprisals[surprisals["token"] != "</s>"]
surprisals.reset_index(drop=True, inplace=True)
```

[10]:
```
find_mismatch(surprisals, sprt)
```

```
Error in:
surprisals_idx: 728, sprt_idx: 12375
surprisals word: guns VS sprt word: guns --
```

Notice the missmatch above is caused since the word “guns” in ‘surprisals’ dataset doesn’t contain

'–'.

Let us further investigate:

[11]:
```
surprisals.loc[[727, 728, 729]]
```

[11]:

	sentence_id	token_id	token	surprisal
727	38	3	three	11.8941
728	38	4	guns	16.7478
729	38	5	--	8.6260

[12]:
```
# remvoing all "--" words from the dataset and check if "--" appears as part of␣
↪a word
r_surprisals = surprisals[surprisals["token"] != "--"]
display(r_surprisals[r_surprisals["token"].str.contains("--")])
print("\n------------------")

# check if "--" appears as an individual word
display(sprt[sprt["word"] == "--"])
print("\n------------------")

# count the number of occurences of "--" in experiment text
print(f'\nnumber of "--" in the experiment text␣
↪{len(surprisals[surprisals["token"] == "--"])}')
```

```
Empty DataFrame
Columns: [sentence_id, token_id, token, surprisal]
Index: []


------------------

Empty DataFrame
Columns: [word, code, subject, text_id, text_pos, word_in_exp, time]
Index: []


------------------

number of "--" in the experiment text 43
```

From the findings above, we can conclude the following:

- In 'surprisals' dataset: "–" only appears as an individual word.
- In 'sprt' dataset: "–" never appears as an individual word, but only as part of a word.

Thus, we can conclude that every time a word of the form "x –" appears in the experiment text, it appears as one word in 'sprt' ("x –") but as two individual words in 'surprisals' ("x", "–").

Since we have individual records for the tokens "x" and "–" (with each of their surprisals values), we can aggregate them to a single record with the token "x –" and a single surprisal value.

After consulting in an office hour regarding the way to perform the aggregation, we were advised to average the surprisals values. Since there're 43 instances of this case, we will write a generic code to implement our solution.

[13]:
```
hyphen_lst = surprisals.index[surprisals["token"] == "--"].tolist()
prev_word_lst = [x-1 for x in hyphen_lst]

surprisals_len = len(surprisals)
for i, row in surprisals.iterrows():

  if i in prev_word_lst:
    avg_surprisal = (row["surprisal"] + surprisals.loc[[i+1]]["surprisal"]) / 2
    combined_word = row["token"] + " --"

    surprisals.at[i, "surprisal"] = avg_surprisal
    surprisals.at[i, "token"] = combined_word

surprisals = surprisals[surprisals["token"] != "--"]
surprisals.reset_index(drop=True, inplace=True)
```

[14]:
```
find_mismatch(surprisals, sprt)
```

```
Error in:
surprisals_idx: 3458, sprt_idx: 64572
surprisals word: and VS sprt word: crossed
```

[15]:
```
surprisals.loc[[i for i in range(3453, 3460)]]
```

[15]:

	sentence_id	token_id	token	surprisal
3453	217	8	the	6.254760
3454	217	9	<unk> --	7.244195
3455	217	11	<unk>	6.898500
3456	217	12	<unk>	5.454490
3457	218	1	<unk>	4.816360
3458	218	2	and	6.818490
3459	218	3	crossed	16.518900

[16]:
```
sprt.loc[[i for i in range(65580, 65650)]]
```

[16]:

	word	code	subject	text_id	text_pos	word_in_exp	time
65580	Jersey	26585	s017	6	585	1252	216.07
65581	Jersey	26585	s005	6	585	3601	354.98
65582	Jersey	26585	s034	6	585	1686	171.56
65583	Jersey	26585	s030	6	585	2555	374.28
65584	Jersey	26585	s015	6	585	4168	326.22
…	…	…	…	…	…	…	…
65645	as	26588	s034	6	588	1689	162.53
65646	as	26588	s013	6	588	1797	265.13

```
65647       as   26588     s021         6         588           2019   542.76
65648       as   26588     s012         6         588           3328   335.19
65649       as   26588     s004         6         588           4117   260.39

[70 rows x 7 columns]
```

After careful examination, we noticed the problem is caused by the word (Reverend! –").

First, we can see that this word contains "–". Therefore it's represented by two differnet records in 'surprisals' (as mentioned previously). However, unlike the prevoius case we dealt with, the token "–" comes with the character (") after it. As a result it's labeld as unkown in 'surprisals'. Therefore our procedure from the previous step didn't work in this case. Hence, we will remove the record manually.

[17]:
```
# remove record 3457 from 'surprisals' [the unknown corresponding to (--")]
surprisals.drop([3457], inplace=True)
surprisals.reset_index(drop=True, inplace=True)
```

[18]:
```
find_mismatch(surprisals, sprt)
```

```
Error in:
surprisals_idx: 4452, sprt_idx: 83781
surprisals word: N. VS sprt word: N. Y.
```

[19]:
```
surprisals.loc[[i for i in range(4450, 4455)]]
```

[19]:

	sentence_id	token_id	token	surprisal
4450	292	15	Port	17.23220
4451	292	16	<unk>	4.77074
4452	292	17	N.	13.81470
4453	292	18	<unk>	2.57398
4454	292	19	But	16.25960

As we can see, the word "N. Y." appears as an individaul word in 'sprt' but as two separate words in 'surprisals'. However, the word "Y." was labeled as "<unk>". Therefore we can't combine the two words into a single record with the token "N. Y." (as we have done before). Thus, in order to deal with this error, we need to delete the records of "N. Y." from 'sprt' and the records of "N." and "<unk>" (corresponds to "Y.") from 'surprisals'.

[21]:
```
# remove all records with the word "N. Y."
sprt = sprt[sprt["word"] != "N. Y."]
sprt.reset_index(drop=True, inplace=True)

# remove records 4452, 4453 from 'surprisals' (corresponds to "N." and "<unk>"␣
↪of "Y.")
surprisals.drop([4452, 4453], inplace=True)
surprisals.reset_index(drop=True, inplace=True)
```

[20]:
```
find_mismatch(surprisals, sprt)
```

```
Error in:
surprisals_idx: 4452, sprt_idx: 83781
surprisals word: N. VS sprt word: N. Y.
```

[22]:
```
surprisals.loc[[4470, 4471]]
```

[22]:

	sentence_id	token_id	token	surprisal
4470	293	12	N.	14.4224
4471	293	13	H.	14.1974

As we can see, the word "N. H." appears as an individaul word in 'sprt' but as two separate words in 'surprisals'. We will act similarly to the hyphen case.

[23]:
```
# find the average surprisal of "N." and "H." and create a new record "N. H."␣
↪accordingly
avg_surprisal = (float(surprisals.loc[[4470]]["surprisal"]) + float(surprisals.
↪loc[[4471]]["surprisal"])) / 2

surprisals.at[4470, "surprisal"] = avg_surprisal
surprisals.at[4470, "token"] = "N. H."
surprisals.drop([4471], inplace=True)
surprisals.reset_index(drop=True, inplace=True)
```

[24]:
```
find_mismatch(surprisals, sprt)
```

```
No errors were found!
```

As we can see, the synchronized order between the datasets 'sprt' and 'surprisals' now holds.

2.2 Harmonize

After we fixed the synchronized order between the two datasets, we can create the harmonized dataset with the following colmuns:

- word.
- surprisal - the surprisal value of the word (from 'surprisals' dataset).
- avg_rt - the average reading time of a word's block across the different subjects (from 'sprt' dataset).

[25]:
```
def harmonize(rt_data: pd.DataFrame, surprs_data: pd.DataFrame) -> pd.DataFrame:

  rows_lst = []
  rt_data_idx, rt_data_len = 0, len(rt_data)

  for _, surprs_row in surprs_data.iterrows():
    sum_time, cnt_time = rt_data.iloc[rt_data_idx]["time"], 1
```

```
    while rt_data_idx < rt_data_len - 1 and \
    rt_data.loc[rt_data_idx]["text_id"] == rt_data.
↪loc[rt_data_idx+1]["text_id"] and \
    rt_data.loc[rt_data_idx]["text_pos"] == rt_data.
↪loc[rt_data_idx+1]["text_pos"]:
      sum_time += rt_data.loc[rt_data_idx+1]["time"]
      cnt_time += 1

      rt_data_idx += 1

    word = surprs_row["token"]
    surprisal = surprs_row["surprisal"]
    avg_rt = sum_time / cnt_time

    rows_lst.append([word, surprisal, avg_rt])
    rt_data_idx += 1

  df = pd.DataFrame(rows_lst, columns=["word", "surprisal", "avg_rt"])
  df = df[~df["word"].str.contains("<unk>")]
  df.reset_index(drop=True, inplace=True)

  return df

harmonized_df = harmonize(sprt, surprisals)
harmonized_df
```

[25]:

	word	surprisal	avg_rt
0	In	4.57937	350.145625
1	County	12.65410	296.042941
2	near	12.22380	348.327500
3	the	1.98095	306.075882
4	River	15.70900	289.048235
…	…	…	…
5494	failed	8.25341	292.772500
5495	as	9.42416	284.470833
5496	a	3.23962	282.622083
5497	leader	12.81650	279.445417
5498	and	5.90348	299.705000

[5499 rows x 3 columns]

When you are done with this step, save the result using the following code

[26]:
```
harmonized_df.to_csv("harmonized_ngram.csv", index=False)
```

Great, now you're ready to start doing analysis on this output data!

3 Analyses

Now that we've obtained our harmonized surprisal-vs-RT files, let's perform some analysis on the data.

[27]:
```
harmonized_df = pd.read_csv("harmonized_ngram.csv")
```

3.1 1. Univariate linear regression

Here is an overview of the analysis we want you to run.

- For each of `metric` in `{surprisal, raw_probability}`:
  - Fit a linear regression model to predict RTs from the metric. You should report the coefficient for the metric term (slope) and a corresponding t-score and p-value (to determine whether it is significantly different from 0), as well as an R^2-score (the coefficient of determination) of the model.;
  - Draw metric-RT scatterplot with best-fit line, **without** binning RT values; and
  - Draw metric-RT scatterplot with best-fit line, **with** binning RT values.

3.1.1 `Metric = Surprisal`

Fit a linear regression

[29]:
```
# Fit and summarize OLS model - surprisal metric
y = harmonized_df["avg_rt"]
X = harmonized_df["surprisal"]

# add 1 column for bias of the regression
X = sm.add_constant(X)

lin_model= sm.OLS(y, X)
lin_model = lin_model.fit()

print(lin_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 avg_rt   R-squared:                       0.040
Model:                            OLS   Adj. R-squared:                  0.039
Method:                 Least Squares   F-statistic:                     226.7
Date:                Sun, 30 Apr 2023   Prob (F-statistic):           3.14e-50
Time:                        16:45:07   Log-Likelihood:                -27809.
No. Observations:                5499   AIC:                         5.562e+04
Df Residuals:                    5497   BIC:                         5.563e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
```

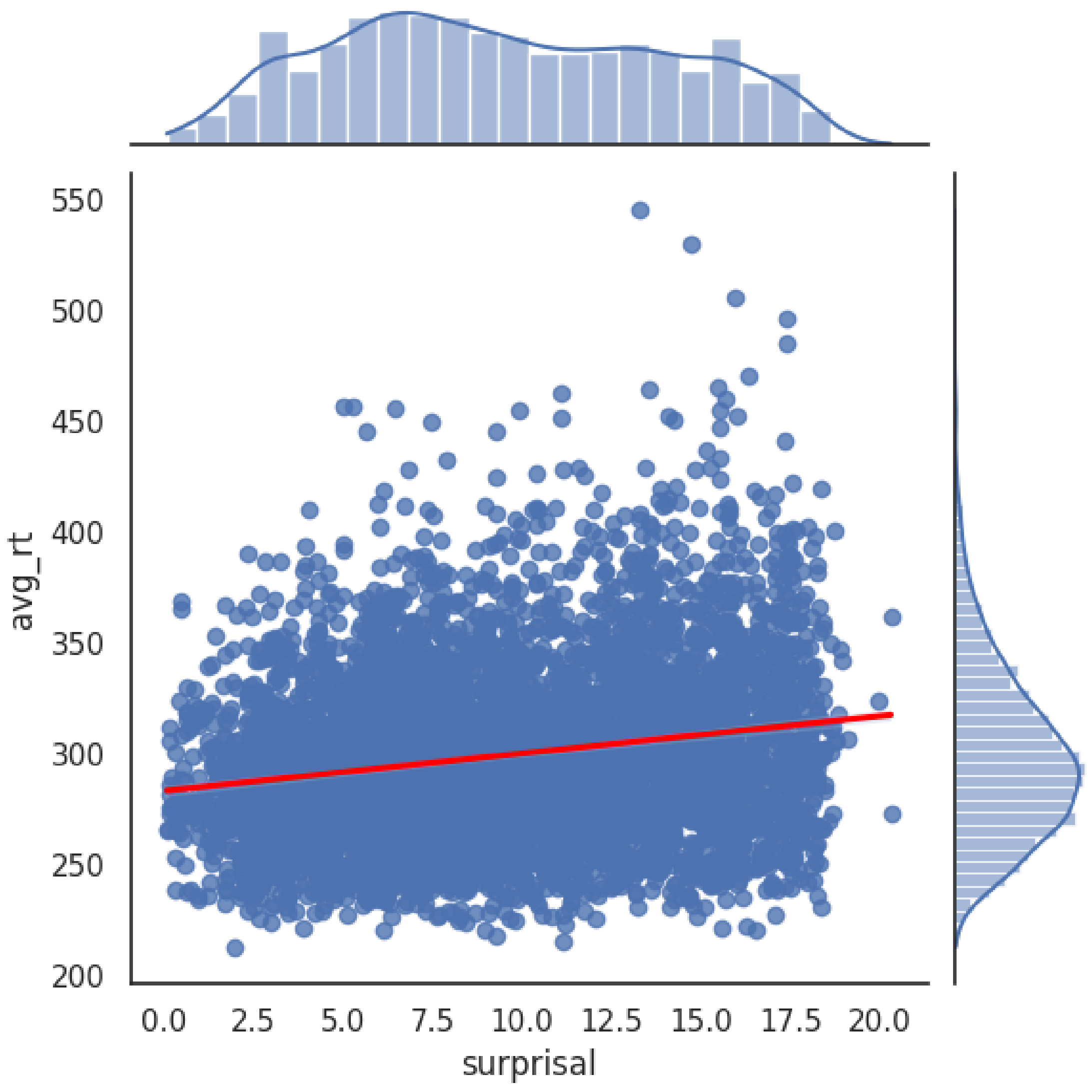
```
const          283.7915      1.174    241.824      0.000     281.491     286.092
surprisal        1.6840      0.112     15.056      0.000       1.465       1.903
==============================================================================
Omnibus:                      630.856   Durbin-Watson:                   1.037
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1033.021
Skew:                           0.804   Prob(JB):                    4.81e-225
Kurtosis:                       4.388   Cond. No.                         24.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

The function `.summary()` outputs a variety of metrices and statistical tests. Here we are intrested in model's parameters (the coefficients), their t score, and the corresponding p-values, as well as in the overall R^2 - score of the model.

Now let's create a scatterplot of our data accompanied by the best-fit line

Without Binning:

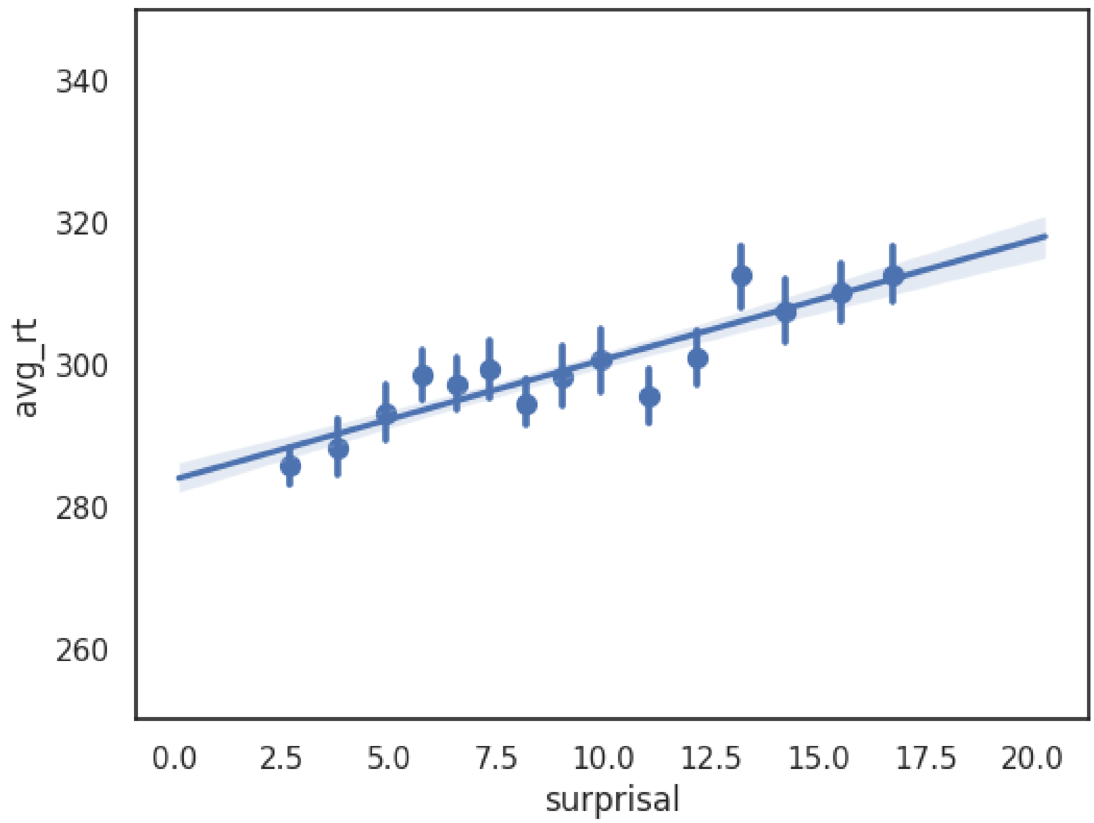
[30]:
```python
g = sns.jointplot(x="surprisal", y="avg_rt", data=harmonized_df, kind='reg')
# We're going to make the regression line red so it's easier to see
regline = g.ax_joint.get_lines()[0]
regline.set_color('red')
```



With Binning:

[31]:
```
g = sns.regplot(x="surprisal", y="avg_rt", data=harmonized_df, x_bins=15)
g.set_ylim([250, 350])
```

[31]: (250.0, 350.0)



3.1.2 Metric = Raw_probability

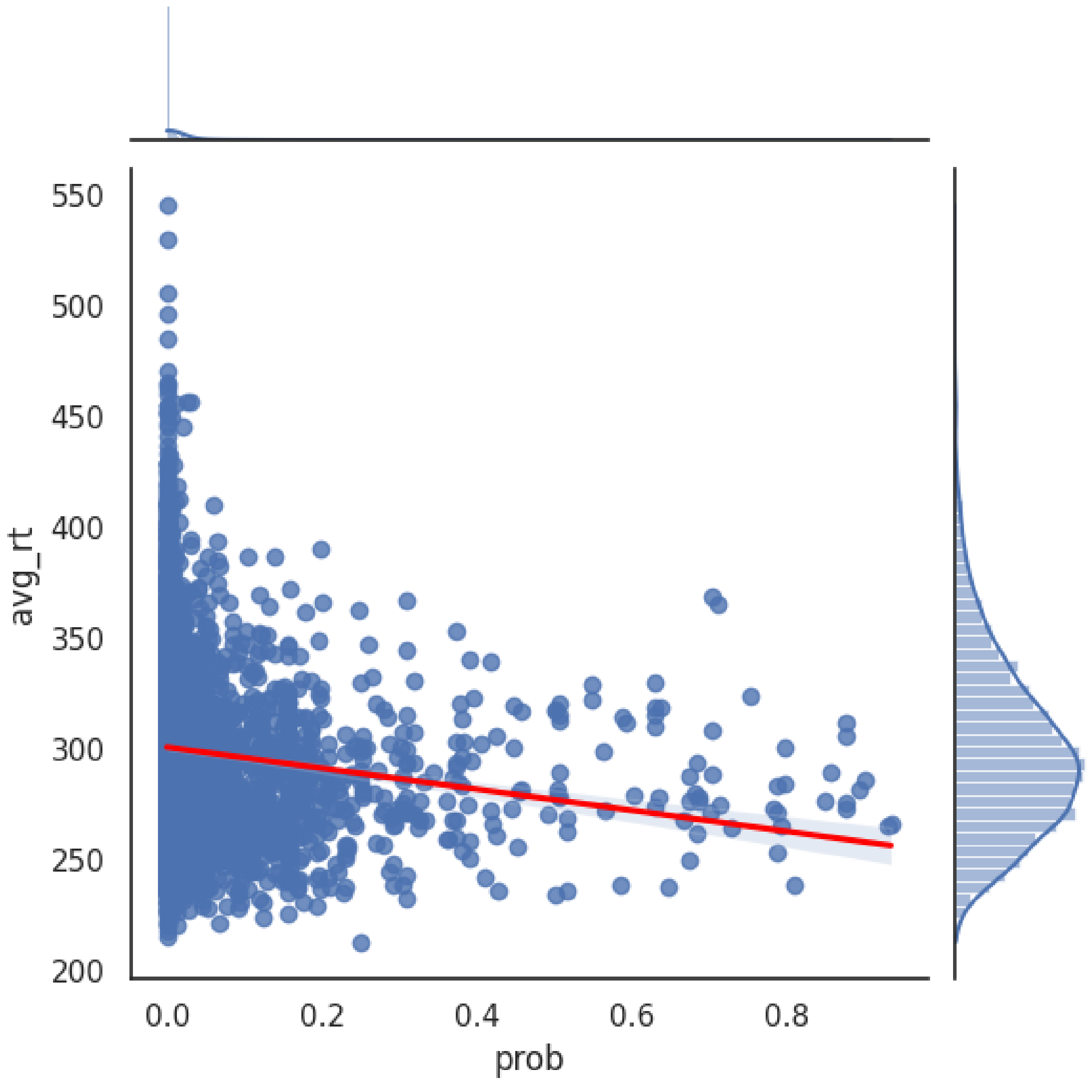
After running the code cells above, your next task is to reproduce this analysis for `metric=raw_probability`.

[34]:
```
# Fit and summarize OLS model - raw probability metric
# we assume the log base in the surprisal calculation is 2 (coures's convention)
raw_prob = np.power(2, -harmonized_df.surprisal)
harmonized_df["prob"] = raw_prob

y = harmonized_df["avg_rt"]
X = harmonized_df["prob"]

# add 1 column for bias of the regression
X = sm.add_constant(X)

lin_model= sm.OLS(y, X)
lin_model = lin_model.fit()

print(lin_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 avg_rt   R-squared:                       0.013
Model:                            OLS   Adj. R-squared:                  0.013
Method:                 Least Squares   F-statistic:                     72.58
Date:                Sun, 30 Apr 2023   Prob (F-statistic):           2.04e-17
Time:                        16:47:29   Log-Likelihood:                -27884.
No. Observations:                5499   AIC:                         5.577e+04
Df Residuals:                    5497   BIC:                         5.578e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        301.2798      0.553    545.261      0.000     300.197     302.363
prob         -47.4723      5.572     -8.520      0.000     -58.396     -36.549
==============================================================================
Omnibus:                      704.873   Durbin-Watson:                   1.023
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1198.527
Skew:                           0.865   Prob(JB):                    5.54e-261
Kurtosis:                       4.497   Cond. No.                         10.7
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

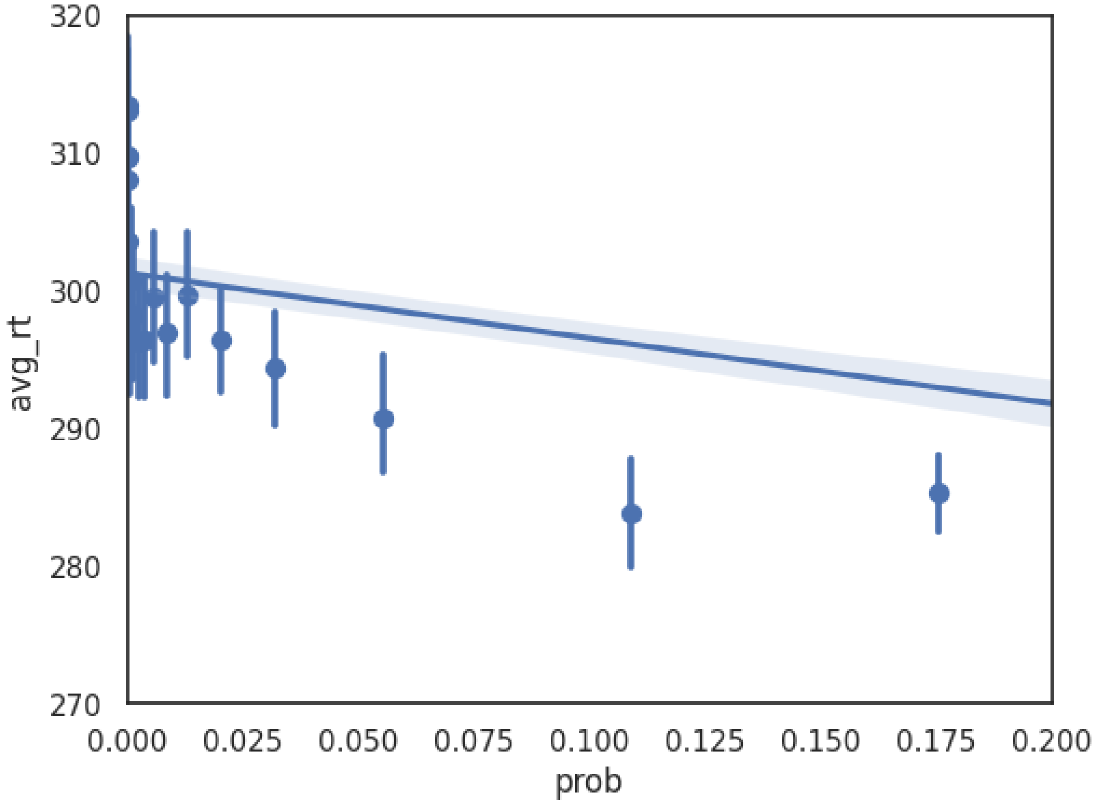
[35]:

```python
g = sns.jointplot(x="prob", y="avg_rt", data=harmonized_df, kind='reg')
# We're going to make the regression line red so it's easier to see
regline = g.ax_joint.get_lines()[0]
regline.set_color('red')
```



[36]:
```
g = sns.regplot(x="prob", y="avg_rt", data=harmonized_df, x_bins=20)
g.set_ylim([270, 320])
g.set_xlim([0, 0.2])
```

[36]: (0.0, 0.2)



3.1.3 Interpret the results

1. Does the univariate analysis support the hypothesis of a linear relationship between word surprisal and word reading time?
2. Is that hypothesis better or worse than an alternative hypothesis of a linear relationship between raw word probability and word reading time?
3. Are there other alternative hypotheses that might be even more compelling given the data?

3.1.4 Our Results Interpetation

1. From the analysis with the metric 'surprisal', we can see that the p-value is approximately 0. This highly indicates statistical significance for the linear relationship between the metric (surprisal) and the mean reading time.

Hence, the analysis supports the hypothesis of a linear relationship between word surprisal and word reading time.

2. From the analysis with the metric 'raw probabilty', we can conclude the following:

- The p-value is also approximately 0, which highly indicates statistical significance for the linear relationship between the metric (raw probabilty) and the mean reading time.
- It appears that $R^2_{prob} = 0.013$, compared to $R^2_{Surp} = 0.040$.

Hence, there is high statistical significance for the alternative hypothesis of a linear relationship between raw word probability and word reading time. However, R^2_{Surp} is 4-times bigger then R^2_{Prob}. This means that the relationship between the mean reading time and the metric 'surprisal' is much stronger than with the metric 'raw probabilty'. Thus, the first hypothesis is better than the alternative one (stronger correlation between the explaining and explained variables).

3. Given the data, it's compelling to check the following univariate hypotheses:

- Is there a correlation between the metric 'word length' and reading time?
- Is there a correlation between the metric 'word frequency' and reading time?
- Is there a correlation between the metric 'number of syllables' and reading time?

Moreover, it's interesting to check a multivariate regression model with the above explaining variables as well as the surprisal (with reading time as explained variable).

3.1.5 2. Multiple regression analysis : Adding control variables

In this stage we want to add two control variables to our linear model and reexamine the effect of surprisal *above and beyond* these variables. The two variables are **word-length** and **word log-frequency**.

First, you should write a code that creates those variables.

Word-length:

[38]:
```
# calculate the word-length for each word in the dataset and add this␣
↪information as a new column in harmonized_ngram.csv
harmonized_df["word_length"] = [len(x) for x in harmonized_df["word"].tolist()]
```

Word log-frequency:

For each word w_i in our `harmonized_ngram.csv` dataset, we want to obtain the $log(frequency(w_i))$ of w_i using a different, large corpus of text. You will first download the *tokenized* version of the **PTB** dataset (no other preprocessing stages are needed) and then write a code for calculating each word's log-frequency.

[39]:
```
# Downloads ptb_tok_train.txt
!wget -qO ptb_tok_train.txt https://gist.githubusercontent.com/scaperex/
↪cdd4231472d6188f03ab21e2b2729fee/raw/
↪e1b4c764561fd038470830534baaa220b0eb4c6d/ptb_tok_train.txt
!head ptb_tok_train.txt
```

```
In an Oct. 19 review of `` The Misanthrope '' at Chicago 's Goodman Theatre
-LRB- `` <unk> <unk> Take the Stage in <unk> City , '' Leisure & Arts -RRB- ,
the role of Celimene , played by Kim <unk> , was mistakenly attributed to
Christina Haag .
Ms. Haag plays <unk> .
Rolls-Royce Motor Cars Inc. said it expects its U.S. sales to remain steady at
about 1,200 cars in 1990 .
The luxury auto maker last year sold <unk> cars in the U.S.
```

```
Howard <unk> , president and chief executive officer , said he anticipates
growth for the luxury auto maker in Britain and Europe , and in Far Eastern
markets .
<unk> INDUSTRIES Inc. increased its quarterly to 10 cents from seven cents a
share .
The new rate will be payable Feb. 15 .
A record date has n't been set .
Bell , based in Los Angeles , makes and distributes electronic , computer and
building products .
Investors are appealing to the Securities and Exchange Commission not to limit
their access to information about stock purchases and sales by corporate
insiders .
```

After examing the given corpus we noticed the following:

- Punctuation marks (’ . ‘,’!’ , ‘?’, etc.) are treated as individual words.
- There isn’t a representation of words of the form “x –” from our dataset (created in the pre-processing stage) in the corpus, but only to “x” and “–” individually (and not consequent).
- There isn’t a representation of the word “N. H.” (created in the pre-processing) from our dataset in the corpus, but only to “N.” and “H.” individually (and not consequent).

After consulting in an office hour we were advied to act as follows:

- Not include any punctuation marks in the frequency calculations.
- Treat the frequnency of the words of the form “X –” as “X” only (since “–” isn’t a real word).
- Remove the record which represetns the word “N. H.” from the dataset (since unlike the previous bullet, both “N.” and “H.” are actual words).

[40]:
```python
# removing the record of the word "N. H." from the dataset
r_harmonized_df = harmonized_df.copy()
r_harmonized_df = r_harmonized_df[r_harmonized_df["word"] != "N. H."]
```

[42]:
```python
#calculate log frequencies and add the information as a new column in harmonized_ngram.csv
with open('ptb_tok_train.txt') as f:
  lines = f.readlines()

word_dict = {}
total_word_num = 0

for line in lines:
  line = line.split(" ")

  for word in line:
    # not including punctuation marks which is treated as words in the calculations
    if word[0] in string.punctuation:
      continue
```

```
    if word in word_dict.keys():
      word_dict[word] += 1

    else:
      word_dict[word] = 1

    total_word_num += 1

log_freq_lst = []

for _, row in r_harmonized_df.iterrows():
  word = row["word"]

  # treat the frequnency of the words of the form "x --" as "x" only
  if "--" in word:
    word = word.split(" ")[0]

  # log base 2 as a convention
  log_word_freq = np.log2(word_dict[word] / total_word_num)
  log_freq_lst.append(log_word_freq)

r_harmonized_df["log_freq"] = log_freq_lst
```

Multiple regression analysis:

Based on the code above (section 1: univariate linear regression), write a new code for multiple regresion analysis.

[43]:
```
# Fit and summarize OLS model - multivariate
X = r_harmonized_df[["surprisal", "word_length", "log_freq"]]
y = r_harmonized_df["avg_rt"]

# add 1 column for bias of the regression
X = sm.add_constant(X)

lin_model= sm.OLS(y, X)
lin_model = lin_model.fit()

print(lin_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 avg_rt   R-squared:                       0.064
Model:                            OLS   Adj. R-squared:                  0.063
Method:                 Least Squares   F-statistic:                     125.1
Date:                Sun, 30 Apr 2023   Prob (F-statistic):           2.17e-78
Time:                        16:50:28   Log-Likelihood:                -27729.
No. Observations:                5498   AIC:                         5.547e+04
```

```
Df Residuals:                    5494   BIC:                         5.549e+04
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        283.9609      1.369    207.494      0.000     281.278     286.644
surprisal      1.9737      0.210      9.421      0.000       1.563       2.384
word_length    3.9103      0.334     11.696      0.000       3.255       4.566
log_freq       1.8231      0.255      7.154      0.000       1.323       2.323
==============================================================================
Omnibus:                      581.161   Durbin-Watson:                   1.040
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              910.288
Skew:                           0.770   Prob(JB):                    2.15e-198
Kurtosis:                       4.266   Cond. No.                         42.8
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

###Interpret the results 1. How does the surprisal coefficient of this model compare to the surprisal coefficient in the univariate model? 2. Does your conclusion regarding the effect of suprisal on RTs from the univariate analysis still hold?

1. **The suprisal coefficient under the multivariate model is $\beta_{multi} = 1.974$ and under the univariate model is $\beta_{uni} = 1.684$.**

We can see that the value of the coefficients is approximately the same. Moreover, its value is roughly the same as the other coefficients in the multivariate model. Therefore, we can conclude that the explainability of surprisal given the other variables (word lentgh, log frequency) remains the same. Thus, surprisal is a good explaining variable in the regression model (with reading time as explained variable).

2. **Our conclusion regarding the effect of surprisal on the reading time from the univarite analysis remains the same.**

As mentioned in section 1, the effect of β_{uni} and β_{multi} on reading time is approximately the same. Thus, our conclusion regading the univarite model still holds.

4 Export to PDF

Run the following cell to download the notebook as a nicely formatted pdf file.

[ ]:
```
# Add to a new cell at the end of the notebook and run the follow code,
# which will save the notebook as pdf in your google drive (allow the␣
↪permissions) and download it automatically.

!wget -nc https://raw.githubusercontent.com/scaperex/colab-pdf/master/colab_pdf.
↪py
```

```python
from colab_pdf import colab_pdf

# If you saved the notebook in the default location in your Google Drive,
#  and didn't change the name of the file, the code should work as is. If not,␣
 ↪adapt accordingly.
# E.g. in your case the file name may be "Copy of XXXX.ipynb"

colab_pdf(file_name='Pset_2_RT_and_surprisal.ipynb', notebookpath="drive/
 ↪MyDrive/Colab Notebooks")
```

```
--2023-04-30 17:36:46--  https://raw.githubusercontent.com/scaperex/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.109.133, 185.199.111.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.109.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 1902 (1.9K) [text/plain]
Saving to: 'colab_pdf.py'

colab_pdf.py        100%[===================>]   1.86K  --.-KB/s    in 0s

2023-04-30 17:36:46 (24.3 MB/s) - 'colab_pdf.py' saved [1902/1902]

Mounted at /content/drive

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```