CPSC 330 - Applied Machine Learning

Homework 8: Word embeddings, time series

Associated lectures: Lectures 16, 18, 19

Due date: Tuesday, June 21, 2022 at 18:00

Table of Contents

In [1]:
```
import os

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.cluster import DBSCAN, KMeans
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import (
    GridSearchCV,
    RandomizedSearchCV,
    cross_validate,
    train_test_split,
)
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler

pd.set_option("display.max_colwidth", 0)
```

Instructions

---

rubric={points:1}

Follow the homework submission instructions.

You may work with a partner on this homework and submit your assignment as a group. Below are some instructions on working as a group.

- The maximum group size is 2.
- Use group work as an opportunity to collaborate and learn new things from each other.
- Be respectful to each other and make sure you understand all the concepts in the assignment well.
- It's your responsibility to make sure that the assignment is submitted by one of the group members before the deadline.
- You can find the instructions on how to do group submission on Gradescope here.

Exercise 1: Exploring pre-trained word embeddings

In lecture 16, we talked about natural language processing (NLP). Using pre-trained word embeddings is very common in NLP. It has been shown that pre-trained word embeddings work well on a variety of text classification tasks. These embeddings are created by training a model like Word2Vec on a huge corpus of text such as a dump of Wikipedia or a dump of the web crawl.

A number of pre-trained word embeddings are available out there. Some popular ones are:

- GloVe
  - trained using the GloVe algorithm
  - published by Stanford University
- fastText pre-trained embeddings for 294 languages
  - trained using the fastText algorithm
  - published by Facebook

In this exercise, you will be exploring GloVe Wikipedia pre-trained embeddings. The code below loads pre-trained word vectors trained on Wikipedia. The vectors are created using an algorithm called GloVe. To run the code, you'll need `gensim` package in your cpsc330 conda environment, which you can install as follows:

```
conda install -n cpsc330 -c anaconda gensim
```

In [2]:
```
import gensim
import gensim.downloader

print("Available models to download:")
print(*gensim.downloader.info()["models"].keys(), sep=", ")
```

```
Available models to download:
fasttext-wiki-news-subwords-300, conceptnet-numberbatch-17-06-300, word2vec-ruscorpora-300, word
2vec-google-news-300, glove-wiki-gigaword-50, glove-wiki-gigaword-100, glove-wiki-gigaword-200,
glove-wiki-gigaword-300, glove-twitter-25, glove-twitter-50, glove-twitter-100, glove-twitter-20
0, __testing_word2vec-matrix-synopsis
```

In [3]:
```
# This will take a while to run when you run it for the first time.
import gensim.downloader as api
```

```
glove_wiki_vectors = api.load("glove-wiki-gigaword-100")
```

In [4]:
```
len(glove_wiki_vectors)
```

Out[4]:
```
400000
```

There are 400,000 word vectors in these pre-trained model.

1.1 Word similarity using pre-trained embeddings

rubric={points:4}

Now that we have GloVe Wiki vectors (`glove_wiki_vectors`) loaded, let's explore the word vectors.

Your tasks:

1. Calculate cosine similarity for the following word pairs (`word_pairs`) using the `similarity` method of the model.
2. Do the similarities make sense?

In [5]:
```
word_pairs = [
    ("coast", "shore"),
    ("clothes", "closet"),
    ("old", "new"),
    ("smart", "intelligent"),
    ("dog", "cat"),
    ("tree", "lawyer"),
]
```

BEGIN SOLUTION

In [6]:
```
for (w1, w2) in word_pairs:
    print(
        "Cosine similarity between %s and %s = %0.3f"
        % (w1, w2, glove_wiki_vectors.similarity(w1, w2))
    )
```

```
Cosine similarity between coast and shore = 0.700
Cosine similarity between clothes and closet = 0.546
Cosine similarity between old and new = 0.643
Cosine similarity between smart and intelligent = 0.755
Cosine similarity between dog and cat = 0.880
Cosine similarity between tree and lawyer = 0.077
```

The similarity scores make sense. The similarity between *coast* and *shore* is on the lower side. This might be the case because these words are not frequently occurring words.

Note the high similarity score between *old* and *new*. When we think of similarity, we usually think of synonyms. But the model is not able to distinguish between synonyms vs. antonyms. When words occur in similar contexts, they are similar according to the model.

END SOLUTION

1.2 Bias in embeddings

rubric={points:10}

Your tasks:

1. In Lecture 16 we saw that our pre-trained word embedding model output an analogy that reinforced a gender stereotype. Give an example of how using such a model could cause harm in the real world.
2. Here we are using pre-trained embeddings which are built using Wikipedia data. Explore whether there are any worrisome biases present in these embeddings or not by trying out some examples. You can use the following two methods or other methods of your choice to explore what kind of stereotypes and biases are encoded in these embeddings.
   - You can use the `analogy` function below which gives words analogies.
   - You can also use similarity or distance methods. (An example is shown below.)
3. Discuss your observations. Do you observe the gender stereotype we observed in class in these embeddings?

> Note that most of the recent embeddings are de-biased. But you might still observe some biases in them. Also, not all stereotypes present in pre-trained embeddings are necessarily bad. But you should be aware of them when you use them in your models.

In [7]:
```
def analogy(word1, word2, word3, model=glove_wiki_vectors):
    """
    Returns analogy word using the given model.

    Parameters
    --------------
    word1 : (str)
        word1 in the analogy relation
    word2 : (str)
        word2 in the analogy relation
    word3 : (str)
        word3 in the analogy relation
    model :
        word embedding model

    Returns
    ---------------
        pd.dataframe
    """
    print("%s : %s :: %s : ?" % (word1, word2, word3))
    sim_words = model.most_similar(positive=[word3, word2], negative=[word1])
    return pd.DataFrame(sim_words, columns=["Analogy word", "Score"])
```

An example of using similarity between words to explore biases and stereotypes.

In [8]:
```
glove_wiki_vectors.similarity("white", "rich")
```

Out[8]:
```
0.447236
```

In [9]:
```
glove_wiki_vectors.similarity("black", "rich")
```

Out[9]:
```
0.51745194
```

BEGIN SOLUTION

Perhaps NLP-based resume screening with such a model could lead to gender bias, say if it was used for hiring a computer programmer, which the model associates with men.

In [10]:
```
analogy("man", "doctor", "woman")
```

```
man : doctor :: woman : ?
```

Out[10]:

	Analogy word	Score
0	nurse	0.773523
1	physician	0.718943
2	doctors	0.682433
3	patient	0.675068
4	dentist	0.672603
5	pregnant	0.664246
6	medical	0.652045
7	nursing	0.645348
8	mother	0.639333
9	hospital	0.638750

In [11]:
```
analogy("man", "intelligent", "woman")
```

```
man : intelligent :: woman : ?
```

Out[11]:

	Analogy word	Score
0	articulate	0.627857
1	thoughtful	0.616820
2	smart	0.587221
3	strong-willed	0.568862
4	vivacious	0.550627
5	opinionated	0.547024
6	perceptive	0.545705
7	inquisitive	0.544539
8	energetic	0.533960
9	literate	0.533894

In [12]:
```
glove_wiki_vectors.similarity("woman", "ugly"), glove_wiki_vectors.similarity(
    "man", "ugly"
)
```

Out[12]:
```
(0.4038871, 0.44594425)
```

In [13]:
```
glove_wiki_vectors.similarity("man", "gentle"), glove_wiki_vectors.similarity(
```

```
    "woman", "gentle"
)
```

Out[13]:
```
(0.40618372, 0.349594)
```

In [14]:
```
glove_wiki_vectors.similarity("indigenous", "smart"), glove_wiki_vectors.similarity(
    "white", "smart"
)
```

Out[14]:
```
(0.05271155, 0.32884747)
```

- The situation seems better than what we observed with the GoogleNews pre-trained embeddings. This is because most of the modern methods to compute word embeddings are debiased.
- That said, we still see some stereotypes such as "white" it much more closer to "smart" than "indigenous" to "smart". That said, the similarity scores also depend upon the frequency of words, and if a certain word does not occur very frequently in the corpus, the similarity scores are going to be low for that word.

END SOLUTION

1.3 Representation of all words in English

rubric={reasoning:1}

Your tasks:

1. The vocabulary size of Wikipedia embeddings is quite large. Do you think it contains **all** words in English language? What would happen if you try to get a word vector that's unlikely to be present in the vocabulary (e.g., the word "cpsc330").

BEGIN SOLUTION

Of course not. The pre-trained twitter embeddings do not contain all words in English language. We see than a number of domain specific words are not present in GloVe twitter embeddings. And this is going to be true for any pre-trained embedding available out there. We get an error when we try to get a word vector for the word "cpsc330".

In [15]:
```
# glove_wiki_vectors['cpsc330']
```

END SOLUTION

1.4 Classification with pre-trained embeddings

rubric={points:8}

In lecture 16, we saw that you can conveniently get word vectors with `spaCy` with `en_core_web_md` model. In this exercise, you'll use word embeddings in multi-class text classification task. We will use HappyDB corpus which contains about 100,000 happy moments classified into 7 categories: *affection, exercise, bonding, nature, leisure, achievement, enjoy_the_moment*. The data was crowd-sourced via Amazon Mechanical Turk. The ground truth label is not available for all examples, and in this homework, we'll only use the examples where ground truth is available (~15,000 examples).

- Download the data from here.
- Unzip the file and copy it in the homework directory.

The code below reads the data CSV (assuming that it's present in the current directory as *cleaned_hm.csv*), cleans it up a bit, and splits it into train and test splits.

Your tasks:

1. Train logistic regression with bag-of-words features and show classification report on the test set.
2. Train logistic regression with average embedding representation extracted using spaCy and show classification report on the test set. (You can find an example of extracting average embedding features using spaCy in lecture 16 under *sentiment classification using average embeddings*.)
3. Discuss your results. Which model is performing well. Which model would be more interpretable?
4. Are you observing any benefits of transfer learning here? Briefly discuss.

In [16]:
```
df = pd.read_csv("cleaned_hm.csv", index_col=0)
sample_df = df.dropna()
sample_df.head()
```

Out[16]:

hmid	wid	reflection_period	original_hm	cleaned_hm	modified	num_sentence	ground_truth_category	predi
27676	206	24h	We had a serious talk with some friends of ours who have been flaky lately. They understood and we had a good evening hanging out.	We had a serious talk with some friends of ours who have been flaky lately. They understood and we had a good evening hanging out.	True	2	bonding	
27678	45	24h	I meditated last night.	I meditated last night.	True	1	leisure	
27697	498	24h	My grandmother start to walk from the bed after a long time.	My grandmother start to walk from the bed after a long time.	True	1	affection	
27705	5732	24h	I picked my daughter up from the airport and we have a fun and good conversation on the way home.	I picked my daughter up from the airport and we have a fun and good conversation on the way home.	True	1	bonding	
27715	2272	24h	when i received flowers from my best friend	when i received flowers from my best friend	True	1	bonding	

In [17]:
```
sample_df = sample_df.rename(
    columns={"cleaned_hm": "moment", "ground_truth_category": "target"}
)
```

In [18]:
```
train_df, test_df = train_test_split(sample_df, test_size=0.3, random_state=123)
X_train, y_train = train_df["moment"], train_df["target"]
X_test, y_test = test_df["moment"], test_df["target"]
```

In [19]:
```
import spacy

nlp = spacy.load("en_core_web_md")
```

BEGIN SOLUTION

In [20]:
```
from sklearn.metrics import classification_report
```

Classification with bag-of-words representation

In [21]:
```
pipe = make_pipeline(
    CountVectorizer(stop_words="english"), LogisticRegression(max_iter=1000)
)
pipe.named_steps["countvectorizer"].fit(X_train)
X_train_transformed = pipe.named_steps["countvectorizer"].transform(X_train)
print("Data matrix shape:", X_train_transformed.shape)
pipe.fit(X_train, y_train);
```

```
Data matrix shape: (9887, 8060)
```

In [22]:
```
print("Train accuracy {:.2f}".format(pipe.score(X_train, y_train)))
print("Test accuracy {:.2f}".format(pipe.score(X_test, y_test)))
```

```
Train accuracy 0.96
Test accuracy 0.82
```

In [23]:
```
print(classification_report(y_test, pipe.predict(X_test)))
```

```
                  precision    recall  f1-score   support

     achievement       0.79      0.87      0.83      1302
       affection       0.90      0.91      0.91      1423
         bonding       0.91      0.85      0.88       492
enjoy_the_moment       0.60      0.54      0.57       469
        exercise       0.91      0.57      0.70        74
         leisure       0.73      0.70      0.71       407
          nature       0.73      0.46      0.57        71

        accuracy                           0.82      4238
       macro avg       0.80      0.70      0.74      4238
    weighted avg       0.82      0.82      0.81      4238
```

In [ ]:
```

```

Classification with average embedding representation

In [24]:
```
X_train_embeddings = pd.DataFrame([text.vector for text in nlp.pipe(X_train)])
X_test_embeddings = pd.DataFrame([text.vector for text in nlp.pipe(X_test)])
```

We have reduced dimensionality from 13,064 to 300!

In [25]:
```
X_train_embeddings.shape
```

Out[25]:
```
(9887, 300)
```

In [26]:
```
X_train_embeddings.head()
```

Out[26]:

	0	1	2	3	4	5	6	7	8	9	...	
0	-0.736316	0.013320	-0.253970	-0.192695	-0.197627	0.083967	0.025311	-0.203508	0.040755	1.971932	...	-0
1	-0.680728	0.204079	-0.075632	-0.186900	-0.140091	-0.008862	0.060424	-0.241370	0.261775	1.655021	...	-0
2	-0.710997	-0.066040	-0.009313	-0.300902	-0.048794	-0.107534	0.032490	-0.325867	0.161800	1.942583	...	-0
3	-0.712573	0.285219	-0.214599	-0.050761	-0.123887	-0.000463	0.108017	-0.212615	0.062140	2.253695	...	-0
4	-0.672972	0.062188	-0.126946	-0.156823	-0.014307	-0.045873	0.048942	-0.339047	0.086747	1.963757	...	-0

5 rows × 300 columns

In [27]:
```
lgr = LogisticRegression(max_iter=1000)
lgr.fit(X_train_embeddings, y_train)
print("Train accuracy {:.2f}".format(lgr.score(X_train_embeddings, y_train)))
print("Test accuracy {:.2f}".format(lgr.score(X_test_embeddings, y_test)))
```

```
Train accuracy 0.73
Test accuracy 0.71
```

In [28]:
```
print(classification_report(y_test, lgr.predict(X_test_embeddings)))
```

```
                  precision    recall  f1-score   support

     achievement       0.73      0.83      0.77      1302
       affection       0.72      0.86      0.79      1423
         bonding       0.65      0.50      0.57       492
enjoy_the_moment       0.56      0.38      0.45       469
        exercise       0.78      0.34      0.47        74
         leisure       0.76      0.56      0.64       407
          nature       0.71      0.42      0.53        71

        accuracy                           0.71      4238
       macro avg       0.70      0.55      0.60      4238
    weighted avg       0.70      0.71      0.70      4238
```

Discussion: Logistic regression with bag of words would be more interpretable as there will be a coefficient associated with each word. The features of average embedding representation do not have a clear human interpretation. Transfer learning is not giving us any gains here. It might be worth trying out pre-trained twitter embeddings rather than Wikipedia embeddings on this dataset.

END SOLUTION

Exercise 2: Exploring time series data

In this exercise we'll be looking at a dataset of avocado prices. You should start by downloading the dataset.

In [29]:
```
df = pd.read_csv("avocado.csv", parse_dates=["Date"], index_col=0)
```

```
df.head()
```

Out[29]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	ye
0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional	201
1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional	201
2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conventional	201
3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conventional	201
4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional	201

In [30]:
```
df.shape
```

Out[30]:
```
(18249, 13)
```

In [31]:
```
df["Date"].min()
```

Out[31]:
```
Timestamp('2015-01-04 00:00:00')
```

In [32]:
```
df["Date"].max()
```

Out[32]:
```
Timestamp('2018-03-25 00:00:00')
```

It looks like the data ranges from the start of 2015 to March 2018 (~3 years ago), for a total of 3.25 years or so. Let's split the data so that we have a 6 months of test data.

In [33]:
```
split_date = "20170925"
train_df = df[df["Date"] <= split_date]
test_df = df[df["Date"] > split_date]
```

In [34]:
```
assert len(train_df) + len(test_df) == len(df)
```

2.1

rubric={points:4}

In the Rain is Australia dataset from lecture, we had different measurements for each Location. What about this dataset: for which categorical feature(s), if any, do we have separate measurements? Justify your answer by referencing the dataset.

BEGIN SOLUTION

In [35]:
```
df.sort_values(by="Date").head()
```

Out[35]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type
51	2015-01-04	1.75	27365.89	9307.34	3844.81	615.28	13598.46	13061.10	537.36	0.0	organic
51	2015-01-04	1.49	17723.17	1189.35	15628.27	0.00	905.55	905.55	0.00	0.0	organic
51	2015-01-04	1.68	2896.72	161.68	206.96	0.00	2528.08	2528.08	0.00	0.0	organic
51	2015-01-04	1.52	54956.80	3013.04	35456.88	1561.70	14925.18	11264.80	3660.38	0.0	conventional
51	2015-01-04	1.64	1505.12	1.27	1129.50	0.00	374.35	186.67	187.68	0.0	organic

From the above, we definitely see measurements on the same day at different regresion. Let's now group by region.

In [36]:
```
df.sort_values(by=["region", "Date"]).head()
```

Out[36]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	ye
51	2015-01-04	1.22	40873.28	2819.50	28287.42	49.90	9716.46	9186.93	529.53	0.0	conventional	20
51	2015-01-04	1.79	1373.95	57.42	153.88	0.00	1162.65	1162.65	0.00	0.0	organic	20
50	2015-01-11	1.24	41195.08	1002.85	31640.34	127.12	8424.77	8036.04	388.73	0.0	conventional	20
50	2015-01-11	1.77	1182.56	39.00	305.12	0.00	838.44	838.44	0.00	0.0	organic	20
49	2015-01-18	1.17	44511.28	914.14	31540.32	135.77	11921.05	11651.09	269.96	0.0	conventional	20

From the above we see that, even in Albany, we have two measurements on the same date. This seems to be due to the type of avocado.

In [37]:
```
df.sort_values(by=["region", "type", "Date"]).head()
```

Out[37]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	ye
51	2015-01-04	1.22	40873.28	2819.50	28287.42	49.90	9716.46	9186.93	529.53	0.0	conventional	20
50	2015-01-11	1.24	41195.08	1002.85	31640.34	127.12	8424.77	8036.04	388.73	0.0	conventional	20
49	2015-01-18	1.17	44511.28	914.14	31540.32	135.77	11921.05	11651.09	269.96	0.0	conventional	20
48	2015-01-25	1.06	45147.50	941.38	33196.16	164.14	10845.82	10103.35	742.47	0.0	conventional	20
47	2015-02-01	0.99	70873.60	1353.90	60017.20	179.32	9323.18	9170.82	152.36	0.0	conventional	20

Great, now we have a sequence of dates with a single row per date. So, the answer is that we have a separate timeseries for each combination of `region` and `type`.

END SOLUTION

2.2

rubric={points:4}

In the Rain in Australia dataset, the measurements were generally equally spaced but with some exceptions. How about with this dataset? Justify your answer by referencing the dataset.

BEGIN SOLUTION

I think it's not unreasonable to do this on `df` rather than `df_train`, but either way is fine.

In [38]:
```
for name, group in df.groupby(["region", "type"]):
    print("%-40s %s" % (name, group["Date"].sort_values().diff().min()))
```

```
('Albany', 'conventional')                7 days 00:00:00
('Albany', 'organic')                     7 days 00:00:00
('Atlanta', 'conventional')               7 days 00:00:00
('Atlanta', 'organic')                    7 days 00:00:00
('BaltimoreWashington', 'conventional')   7 days 00:00:00
('BaltimoreWashington', 'organic')        7 days 00:00:00
('Boise', 'conventional')                 7 days 00:00:00
('Boise', 'organic')                      7 days 00:00:00
('Boston', 'conventional')                7 days 00:00:00
('Boston', 'organic')                     7 days 00:00:00
('BuffaloRochester', 'conventional')      7 days 00:00:00
('BuffaloRochester', 'organic')           7 days 00:00:00
('California', 'conventional')            7 days 00:00:00
('California', 'organic')                 7 days 00:00:00
('Charlotte', 'conventional')             7 days 00:00:00
('Charlotte', 'organic')                  7 days 00:00:00
('Chicago', 'conventional')               7 days 00:00:00
('Chicago', 'organic')                    7 days 00:00:00
('CincinnatiDayton', 'conventional')      7 days 00:00:00
('CincinnatiDayton', 'organic')           7 days 00:00:00
('Columbus', 'conventional')              7 days 00:00:00
('Columbus', 'organic')                   7 days 00:00:00
('DallasFtWorth', 'conventional')         7 days 00:00:00
('DallasFtWorth', 'organic')              7 days 00:00:00
('Denver', 'conventional')                7 days 00:00:00
('Denver', 'organic')                     7 days 00:00:00
('Detroit', 'conventional')               7 days 00:00:00
('Detroit', 'organic')                    7 days 00:00:00
('GrandRapids', 'conventional')           7 days 00:00:00
('GrandRapids', 'organic')                7 days 00:00:00
('GreatLakes', 'conventional')            7 days 00:00:00
('GreatLakes', 'organic')                 7 days 00:00:00
('HarrisburgScranton', 'conventional')    7 days 00:00:00
('HarrisburgScranton', 'organic')         7 days 00:00:00
('HartfordSpringfield', 'conventional')   7 days 00:00:00
('HartfordSpringfield', 'organic')        7 days 00:00:00
('Houston', 'conventional')               7 days 00:00:00
('Houston', 'organic')                    7 days 00:00:00
('Indianapolis', 'conventional')          7 days 00:00:00
('Indianapolis', 'organic')               7 days 00:00:00
('Jacksonville', 'conventional')          7 days 00:00:00
('Jacksonville', 'organic')               7 days 00:00:00
('LasVegas', 'conventional')              7 days 00:00:00
('LasVegas', 'organic')                   7 days 00:00:00
('LosAngeles', 'conventional')            7 days 00:00:00
('LosAngeles', 'organic')                 7 days 00:00:00
('Louisville', 'conventional')            7 days 00:00:00
('Louisville', 'organic')                 7 days 00:00:00
('MiamiFtLauderdale', 'conventional')     7 days 00:00:00
('MiamiFtLauderdale', 'organic')          7 days 00:00:00
('Midsouth', 'conventional')              7 days 00:00:00
('Midsouth', 'organic')                   7 days 00:00:00
('Nashville', 'conventional')             7 days 00:00:00
('Nashville', 'organic')                  7 days 00:00:00
('NewOrleansMobile', 'conventional')      7 days 00:00:00
('NewOrleansMobile', 'organic')           7 days 00:00:00
('NewYork', 'conventional')               7 days 00:00:00
('NewYork', 'organic')                    7 days 00:00:00
('Northeast', 'conventional')             7 days 00:00:00
('Northeast', 'organic')                  7 days 00:00:00
('NorthernNewEngland', 'conventional')    7 days 00:00:00
('NorthernNewEngland', 'organic')         7 days 00:00:00
('Orlando', 'conventional')               7 days 00:00:00
('Orlando', 'organic')                    7 days 00:00:00
```

```
('Philadelphia', 'conventional')         7 days 00:00:00
('Philadelphia', 'organic')              7 days 00:00:00
('PhoenixTucson', 'conventional')        7 days 00:00:00
('PhoenixTucson', 'organic')             7 days 00:00:00
('Pittsburgh', 'conventional')           7 days 00:00:00
('Pittsburgh', 'organic')                7 days 00:00:00
('Plains', 'conventional')               7 days 00:00:00
('Plains', 'organic')                    7 days 00:00:00
('Portland', 'conventional')             7 days 00:00:00
('Portland', 'organic')                  7 days 00:00:00
('RaleighGreensboro', 'conventional')    7 days 00:00:00
('RaleighGreensboro', 'organic')         7 days 00:00:00
('RichmondNorfolk', 'conventional')      7 days 00:00:00
('RichmondNorfolk', 'organic')           7 days 00:00:00
('Roanoke', 'conventional')              7 days 00:00:00
('Roanoke', 'organic')                   7 days 00:00:00
('Sacramento', 'conventional')           7 days 00:00:00
('Sacramento', 'organic')                7 days 00:00:00
('SanDiego', 'conventional')             7 days 00:00:00
('SanDiego', 'organic')                  7 days 00:00:00
('SanFrancisco', 'conventional')         7 days 00:00:00
('SanFrancisco', 'organic')              7 days 00:00:00
('Seattle', 'conventional')              7 days 00:00:00
('Seattle', 'organic')                   7 days 00:00:00
('SouthCarolina', 'conventional')        7 days 00:00:00
('SouthCarolina', 'organic')             7 days 00:00:00
('SouthCentral', 'conventional')         7 days 00:00:00
('SouthCentral', 'organic')              7 days 00:00:00
('Southeast', 'conventional')            7 days 00:00:00
('Southeast', 'organic')                 7 days 00:00:00
('Spokane', 'conventional')              7 days 00:00:00
('Spokane', 'organic')                   7 days 00:00:00
('StLouis', 'conventional')              7 days 00:00:00
('StLouis', 'organic')                   7 days 00:00:00
('Syracuse', 'conventional')             7 days 00:00:00
('Syracuse', 'organic')                  7 days 00:00:00
('Tampa', 'conventional')                7 days 00:00:00
('Tampa', 'organic')                     7 days 00:00:00
('TotalUS', 'conventional')              7 days 00:00:00
('TotalUS', 'organic')                   7 days 00:00:00
('West', 'conventional')                 7 days 00:00:00
('West', 'organic')                      7 days 00:00:00
('WestTexNewMexico', 'conventional')     7 days 00:00:00
('WestTexNewMexico', 'organic')          7 days 00:00:00
```

In [39]:

```
for name, group in df.groupby(["region", "type"]):
    print("%-40s %s" % (name, group["Date"].sort_values().diff().max()))
```

```
('Albany', 'conventional')                7 days 00:00:00
('Albany', 'organic')                     7 days 00:00:00
('Atlanta', 'conventional')               7 days 00:00:00
('Atlanta', 'organic')                    7 days 00:00:00
('BaltimoreWashington', 'conventional')   7 days 00:00:00
('BaltimoreWashington', 'organic')        7 days 00:00:00
('Boise', 'conventional')                 7 days 00:00:00
('Boise', 'organic')                      7 days 00:00:00
('Boston', 'conventional')                7 days 00:00:00
('Boston', 'organic')                     7 days 00:00:00
('BuffaloRochester', 'conventional')      7 days 00:00:00
('BuffaloRochester', 'organic')           7 days 00:00:00
('California', 'conventional')            7 days 00:00:00
('California', 'organic')                 7 days 00:00:00
('Charlotte', 'conventional')             7 days 00:00:00
('Charlotte', 'organic')                  7 days 00:00:00
('Chicago', 'conventional')               7 days 00:00:00
('Chicago', 'organic')                    7 days 00:00:00
('CincinnatiDayton', 'conventional')      7 days 00:00:00
('CincinnatiDayton', 'organic')           7 days 00:00:00
('Columbus', 'conventional')              7 days 00:00:00
('Columbus', 'organic')                   7 days 00:00:00
('DallasFtWorth', 'conventional')         7 days 00:00:00
('DallasFtWorth', 'organic')              7 days 00:00:00
('Denver', 'conventional')                7 days 00:00:00
('Denver', 'organic')                     7 days 00:00:00
('Detroit', 'conventional')               7 days 00:00:00
('Detroit', 'organic')                    7 days 00:00:00
('GrandRapids', 'conventional')           7 days 00:00:00
('GrandRapids', 'organic')                7 days 00:00:00
('GreatLakes', 'conventional')            7 days 00:00:00
('GreatLakes', 'organic')                 7 days 00:00:00
('HarrisburgScranton', 'conventional')    7 days 00:00:00
('HarrisburgScranton', 'organic')         7 days 00:00:00
('HartfordSpringfield', 'conventional')   7 days 00:00:00
('HartfordSpringfield', 'organic')        7 days 00:00:00
('Houston', 'conventional')               7 days 00:00:00
('Houston', 'organic')                    7 days 00:00:00
('Indianapolis', 'conventional')          7 days 00:00:00
('Indianapolis', 'organic')               7 days 00:00:00
('Jacksonville', 'conventional')          7 days 00:00:00
('Jacksonville', 'organic')               7 days 00:00:00
('LasVegas', 'conventional')              7 days 00:00:00
('LasVegas', 'organic')                   7 days 00:00:00
('LosAngeles', 'conventional')            7 days 00:00:00
('LosAngeles', 'organic')                 7 days 00:00:00
('Louisville', 'conventional')            7 days 00:00:00
('Louisville', 'organic')                 7 days 00:00:00
('MiamiFtLauderdale', 'conventional')     7 days 00:00:00
('MiamiFtLauderdale', 'organic')          7 days 00:00:00
('Midsouth', 'conventional')              7 days 00:00:00
('Midsouth', 'organic')                   7 days 00:00:00
('Nashville', 'conventional')             7 days 00:00:00
('Nashville', 'organic')                  7 days 00:00:00
('NewOrleansMobile', 'conventional')      7 days 00:00:00
('NewOrleansMobile', 'organic')           7 days 00:00:00
('NewYork', 'conventional')               7 days 00:00:00
('NewYork', 'organic')                    7 days 00:00:00
('Northeast', 'conventional')             7 days 00:00:00
('Northeast', 'organic')                  7 days 00:00:00
('NorthernNewEngland', 'conventional')    7 days 00:00:00
('NorthernNewEngland', 'organic')         7 days 00:00:00
('Orlando', 'conventional')               7 days 00:00:00
('Orlando', 'organic')                    7 days 00:00:00
```

```
('Philadelphia', 'conventional')         7 days 00:00:00
('Philadelphia', 'organic')              7 days 00:00:00
('PhoenixTucson', 'conventional')        7 days 00:00:00
('PhoenixTucson', 'organic')             7 days 00:00:00
('Pittsburgh', 'conventional')           7 days 00:00:00
('Pittsburgh', 'organic')                7 days 00:00:00
('Plains', 'conventional')               7 days 00:00:00
('Plains', 'organic')                    7 days 00:00:00
('Portland', 'conventional')             7 days 00:00:00
('Portland', 'organic')                  7 days 00:00:00
('RaleighGreensboro', 'conventional')    7 days 00:00:00
('RaleighGreensboro', 'organic')         7 days 00:00:00
('RichmondNorfolk', 'conventional')      7 days 00:00:00
('RichmondNorfolk', 'organic')           7 days 00:00:00
('Roanoke', 'conventional')              7 days 00:00:00
('Roanoke', 'organic')                   7 days 00:00:00
('Sacramento', 'conventional')           7 days 00:00:00
('Sacramento', 'organic')                7 days 00:00:00
('SanDiego', 'conventional')             7 days 00:00:00
('SanDiego', 'organic')                  7 days 00:00:00
('SanFrancisco', 'conventional')         7 days 00:00:00
('SanFrancisco', 'organic')              7 days 00:00:00
('Seattle', 'conventional')              7 days 00:00:00
('Seattle', 'organic')                   7 days 00:00:00
('SouthCarolina', 'conventional')        7 days 00:00:00
('SouthCarolina', 'organic')             7 days 00:00:00
('SouthCentral', 'conventional')         7 days 00:00:00
('SouthCentral', 'organic')              7 days 00:00:00
('Southeast', 'conventional')            7 days 00:00:00
('Southeast', 'organic')                 7 days 00:00:00
('Spokane', 'conventional')              7 days 00:00:00
('Spokane', 'organic')                   7 days 00:00:00
('StLouis', 'conventional')              7 days 00:00:00
('StLouis', 'organic')                   7 days 00:00:00
('Syracuse', 'conventional')             7 days 00:00:00
('Syracuse', 'organic')                  7 days 00:00:00
('Tampa', 'conventional')                7 days 00:00:00
('Tampa', 'organic')                     7 days 00:00:00
('TotalUS', 'conventional')              7 days 00:00:00
('TotalUS', 'organic')                   7 days 00:00:00
('West', 'conventional')                 7 days 00:00:00
('West', 'organic')                      7 days 00:00:00
('WestTexNewMexico', 'conventional')     7 days 00:00:00
('WestTexNewMexico', 'organic')         21 days 00:00:00
```

It looks almost perfect - just organic avocados in WestTexNewMexico seems to be missing a couple measurements.

In [40]:
```
name
```

Out[40]:
```
('WestTexNewMexico', 'organic')
```

In [41]:
```
group["Date"].sort_values().diff().value_counts()
```

Out[41]:
```
7 days     163
14 days    1
21 days    1
Name: Date, dtype: int64
```

So, in one case there's a 2-week jump, and in one cast there's a 3-week jump.

In [42]:
```
group["Date"].sort_values().reset_index(drop=True).diff().sort_values()
```

Out[42]:
```
1      7 days
106    7 days
107    7 days
108    7 days
109    7 days
        ...
52     7 days
165    7 days
48    14 days
127   21 days
0        NaT
Name: Date, Length: 166, dtype: timedelta64[ns]
```

We can see the anomalies occur at index 48 and 127. (Note: I had to `reset_index` because the index was not unique to each row.)

In [43]:
```
group["Date"].sort_values().reset_index(drop=True)[45:50]
```

Out[43]:
```
45   2015-11-15
46   2015-11-22
47   2015-11-29
48   2015-12-13
49   2015-12-20
Name: Date, dtype: datetime64[ns]
```

We can spot the first anomaly: a 2-week jump from Nov 29, 2015 to Dec 13, 2015.

In [44]:
```
group["Date"].sort_values().reset_index(drop=True)[125:130]
```

Out[44]:
```
125   2017-06-04
126   2017-06-11
127   2017-07-02
128   2017-07-09
129   2017-07-16
Name: Date, dtype: datetime64[ns]
```

And we can spot the second anomaly: a 3-week jump from June 11, 2017 to July 2, 2017.

END SOLUTION

2.3

rubric={points:4}

In the Rain is Australia dataset, each location was a different place in Australia. For this dataset, look at the names of the regions. Do you think the regions are all distinct, or are there overlapping regions? Justify your answer by referencing the data.

BEGIN SOLUTION

In [45]:
```
df["region"].unique()
```

Out[45]:
```
array(['Albany', 'Atlanta', 'BaltimoreWashington', 'Boise', 'Boston',
       'BuffaloRochester', 'California', 'Charlotte', 'Chicago',
       'CincinnatiDayton', 'Columbus', 'DallasFtWorth', 'Denver',
       'Detroit', 'GrandRapids', 'GreatLakes', 'HarrisburgScranton',
       'HartfordSpringfield', 'Houston', 'Indianapolis', 'Jacksonville',
       'LasVegas', 'LosAngeles', 'Louisville', 'MiamiFtLauderdale',
       'Midsouth', 'Nashville', 'NewOrleansMobile', 'NewYork',
       'Northeast', 'NorthernNewEngland', 'Orlando', 'Philadelphia',
       'PhoenixTucson', 'Pittsburgh', 'Plains', 'Portland',
       'RaleighGreensboro', 'RichmondNorfolk', 'Roanoke', 'Sacramento',
       'SanDiego', 'SanFrancisco', 'Seattle', 'SouthCarolina',
       'SouthCentral', 'Southeast', 'Spokane', 'StLouis', 'Syracuse',
       'Tampa', 'TotalUS', 'West', 'WestTexNewMexico'], dtype=object)
```

There seems to be a hierarchical structure here: `TotalUS` is split into bigger regions like `West`, `Southeast`, `Northeast`, `Midsouth`; and `California` is split into cities like `Sacramento`, `SanDiego`, `LosAngeles`. It's a bit hard to figure out what's going on.

In [46]:
```
df.query("region == 'TotalUS' and type == 'conventional' and Date == '20150104'")[
    "Total Volume"
].values[0]
```

Out[46]:
```
31324277.73
```

In [47]:
```
df.query("region != 'TotalUS' and type == 'conventional' and Date == '20150104'")[
    "Total Volume"
].sum()
```

Out[47]:
```
51730521.73
```

Since the individual regions sum up to more than the total US, it seems that some of the other regions are double-counted, which is consistent with a hierarchical structure. For example, Los Angeles is probalby double counted because it's within `LosAngeles` but also within `California`. What a mess!

END SOLUTION

We will use the entire dataset despite any location-based weirdness uncovered in the previous part.

We would like to forecast the avocado price, which is the `AveragePrice` column. The function below is adapted from Lecture 18, with some improvements.

In [48]:
```
def create_lag_feature(
    df, orig_feature, lag, groupby, new_feature_name=None, clip=False
):
    """
    Creates a new feature that's a lagged version of an existing one.

    NOTE: assumes df is already sorted by the time columns and has unique indices.

    Parameters
    ----------
    df : pandas.core.frame.DataFrame
        The dataset.
    orig_feature : str
        The column name of the feature we're copying
```

```
    lag : int
        The lag; negative lag means values from the past, positive lag means values from the fut
    groupby : list
        Column(s) to group by in case df contains multiple time series
    new_feature_name : str
        Override the default name of the newly created column
    clip : bool
        If True, remove rows with a NaN values for the new feature

    Returns
    -------
    pandas.core.frame.DataFrame
        A new dataframe with the additional column added.

    TODO: could/should simplify this function by using `df.shift()`
    """

    if new_feature_name is None:
        if lag < 0:
            new_feature_name = "%s_lag%d" % (orig_feature, -lag)
        else:
            new_feature_name = "%s_ahead%d" % (orig_feature, lag)

    new_df = df.assign(**{new_feature_name: np.nan})
    for name, group in new_df.groupby(groupby):
        if lag < 0:  # take values from the past
            new_df.loc[group.index[-lag:], new_feature_name] = group.iloc[:lag][
                orig_feature
            ].values
        else:  # take values from the future
            new_df.loc[group.index[:-lag], new_feature_name] = group.iloc[lag:][
                orig_feature
            ].values

    if clip:
        new_df = new_df.dropna(subset=[new_feature_name])

    return new_df
```

We first sort our dataframe properly:

In [49]:
```
df_sort = df.sort_values(by=["region", "type", "Date"]).reset_index(drop=True)
df_sort
```

Out[49]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type
0	2015-01-04	1.22	40873.28	2819.50	28287.42	49.90	9716.46	9186.93	529.53	0.0	conventional
1	2015-01-11	1.24	41195.08	1002.85	31640.34	127.12	8424.77	8036.04	388.73	0.0	conventional
2	2015-01-18	1.17	44511.28	914.14	31540.32	135.77	11921.05	11651.09	269.96	0.0	conventional
3	2015-01-25	1.06	45147.50	941.38	33196.16	164.14	10845.82	10103.35	742.47	0.0	conventional
4	2015-02-01	0.99	70873.60	1353.90	60017.20	179.32	9323.18	9170.82	152.36	0.0	conventional
...	...	...	...	...	...	...	...	...	...	...	...
18244	2018-02-25	1.57	18421.24	1974.26	2482.65	0.00	13964.33	13698.27	266.06	0.0	organic
18245	2018-03-04	1.54	17393.30	1832.24	1905.57	0.00	13655.49	13401.93	253.56	0.0	organic
18246	2018-03-11	1.56	22128.42	2162.67	3194.25	8.93	16762.57	16510.32	252.25	0.0	organic
18247	2018-03-18	1.56	15896.38	2055.35	1499.55	0.00	12341.48	12114.81	226.67	0.0	organic
18248	2018-03-25	1.62	15303.40	2325.30	2171.66	0.00	10806.44	10569.80	236.64	0.0	organic

18249 rows × 13 columns

We then call `create_lag_feature`. This creates a new column in the dataset `AveragePriceNextWeek`, which is the following week's `AveragePrice`. We have set `clip=True` which means it will remove rows where the target would be missing.

In [50]:
```
df_hastarget = create_lag_feature(
    df_sort, "AveragePrice", +1, ["region", "type"], "AveragePriceNextWeek", clip=True
)
df_hastarget
```

Out[50]:

	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type
0	2015-01-04	1.22	40873.28	2819.50	28287.42	49.90	9716.46	9186.93	529.53	0.0	conventional
1	2015-01-11	1.24	41195.08	1002.85	31640.34	127.12	8424.77	8036.04	388.73	0.0	conventional
2	2015-01-18	1.17	44511.28	914.14	31540.32	135.77	11921.05	11651.09	269.96	0.0	conventional
3	2015-01-25	1.06	45147.50	941.38	33196.16	164.14	10845.82	10103.35	742.47	0.0	conventional
4	2015-02-01	0.99	70873.60	1353.90	60017.20	179.32	9323.18	9170.82	152.36	0.0	conventional
...	...	...	...	...	...	...	...	...	...	...	...
18243	2018-02-18	1.56	17597.12	1892.05	1928.36	0.00	13776.71	13553.53	223.18	0.0	organic
18244	2018-02-25	1.57	18421.24	1974.26	2482.65	0.00	13964.33	13698.27	266.06	0.0	organic
18245	2018-03-04	1.54	17393.30	1832.24	1905.57	0.00	13655.49	13401.93	253.56	0.0	organic
18246	2018-03-11	1.56	22128.42	2162.67	3194.25	8.93	16762.57	16510.32	252.25	0.0	organic
18247	2018-03-18	1.56	15896.38	2055.35	1499.55	0.00	12341.48	12114.81	226.67	0.0	organic

18141 rows × 14 columns

I will now split the data:

In [51]:
```
train_df = df_hastarget[df_hastarget["Date"] <= split_date]
test_df = df_hastarget[df_hastarget["Date"] > split_date]
```

2.4 Baseline

rubric={points:4}

Let's try a baseline. Previously we used `DummyClassifier` or `DummyRegressor` as a baseline. This time, we'll do something else as a baseline: we'll assume the price stays the same from this week to next week. So, we'll set our prediction of "AveragePriceNextWeek" exactly equal to "AveragePrice", assuming no change. That is kind of like saying, "If it's raining today then I'm guessing it will be raining tomorrow". This simplistic approach will not get a great score but it's a good starting point for reference. If our model does worse that this, it must not be very good.

Using this baseline approach, what R^2 do you get?

BEGIN SOLUTION

In [52]:
```
train_df.columns
```

Out[52]:
```
Index(['Date', 'AveragePrice', 'Total Volume', '4046', '4225', '4770',
       'Total Bags', 'Small Bags', 'Large Bags', 'XLarge Bags', 'type', 'year',
       'region', 'AveragePriceNextWeek'],
      dtype='object')
```

In [53]:
```
r2_score(train_df["AveragePriceNextWeek"], train_df["AveragePrice"])
```

Out[53]:
```
0.8285800937261841
```

In [54]:
```
r2_score(test_df["AveragePriceNextWeek"], test_df["AveragePrice"])
```

Out[54]:
```
0.7631780188583048
```

In [55]:
```
test_df["AveragePrice"]
```

Out[55]:
```
143      1.69
144      1.78
145      1.65
146      1.56
147      1.67
         ...
18243    1.56
18244    1.57
18245    1.54
18246    1.56
18247    1.56
Name: AveragePrice, Length: 2700, dtype: float64
```

In [56]:
```
test_df["AveragePriceNextWeek"]
```

Out[56]:
```
143      1.78
144      1.65
145      1.56
146      1.67
147      1.62
         ...
18243    1.57
18244    1.54
18245    1.56
18246    1.56
18247    1.62
Name: AveragePriceNextWeek, Length: 2700, dtype: float64
```

Interesting that this is a less effective prediction strategy in the later part of the dataset. I guess that means the price was fluctuating more in late 2017 / early 2018?

END SOLUTION

(Optional) 2.5 Modeling

rubric={points:2}

Now that the baseline is done, let's build some models to forecast the average avocado price a week later. Experiment with a few approachs for encoding the date. Justify the decisions you make. Which approach

worked best? Report your test score and briefly discuss your results.

> Because we only have 2 splits here, we need to be a bit wary of overfitting on the test set. Try not to test on it a ridiculous number of times. If you are interested in some proper ways of dealing with this, see for example sklearn's TimeSeriesSplit, which is like cross-validation for time series data.

Exercise 3: Short answer questions

Each question is worth 2 points.

3.1

rubric={points:4}

The following questions pertain to Lecture 18 on time series data:

1. Sometimes a time series has missing time points or, worse, time points that are unequally spaced in general. Give an example of a real world situation where the time series data would have unequally spaced time points.
2. In class we discussed two approaches to using temporal information: encoding the date as one or more features, and creating lagged versions of features. Which of these (one/other/both/neither) two approaches would struggle with unequally spaced time points? Briefly justify your answer.

BEGIN SOLUTION

1. Many many examples: credit card transactions, log files, basically any situation where the frequency of the measurements could not be chosen by the person taking the measurements.
2. Encoding the date as, e.g. OHE month works just fine with unequally spaced points. However, the lag features are more problematic, because the "previous" measurement will be a different length of time away in each case.

END SOLUTION

3.2

rubric={points:6}

The following questions pertain to Lecture 19 on survival analysis. We'll consider the use case of customer churn analysis.

1. What is the problem with simply labeling customers are "churned" or "not churned" and using standard supervised learning techniques, as we did in hw5?
2. Consider customer A who just joined last week vs. customer B who has been with the service for a year. Who do you expect will leave the service first: probably customer A, probably customer B, or we don't have enough information to answer?
3. If a customer's survival function is almost flat during a certain period, how do we interpret that?

BEGIN SOLUTION

1. The "not churned" are censored - we don't know if they will churn shortly or in a long time. These people have the same label and our model will be impacted negatively.
2. Not enough information - it depends! Imagine a subscription service where you have to pay a starter fee after a month and then pay a huge fee after a year. Well, customer B just paid that huge fee and will probably stay a while, whereas customer A may leave before paying the huge fee, so customer A will probably leave first. But imagine a service where people are more and more likely to leave every day, e.g. a movie service with only 100 movies, so you can run out easily. In that case customer B will probably leave first.
3. The customer is very unlikely to leave during that period.

END SOLUTION

(Optional) Exercise 4

rubric={points:1}

Your tasks:

What is your biggest takeaway from this course?

> I'm looking forward to read your answers.

Submission instructions

PLEASE READ: When you are ready to submit your assignment do the following:

1. Run all cells in your notebook to make sure there are no errors by doing `Kernel -> Restart Kernel and Clear All Outputs` and then `Run -> Run All Cells`.

2. Notebooks with cell execution numbers out of order or not starting from “1” will have marks deducted. Notebooks without the output displayed may not be graded at all (because we need to see the output in order to grade your work).
3. Upload the assignment using Gradescope's drag and drop tool. Check out this Gradescope Student Guide if you need help with Gradescope submission.

Congratulations on finishing all homework assignments!

In [57]:
```
from IPython.display import Image

Image("eva-congrats.png")
```

Out[57]: