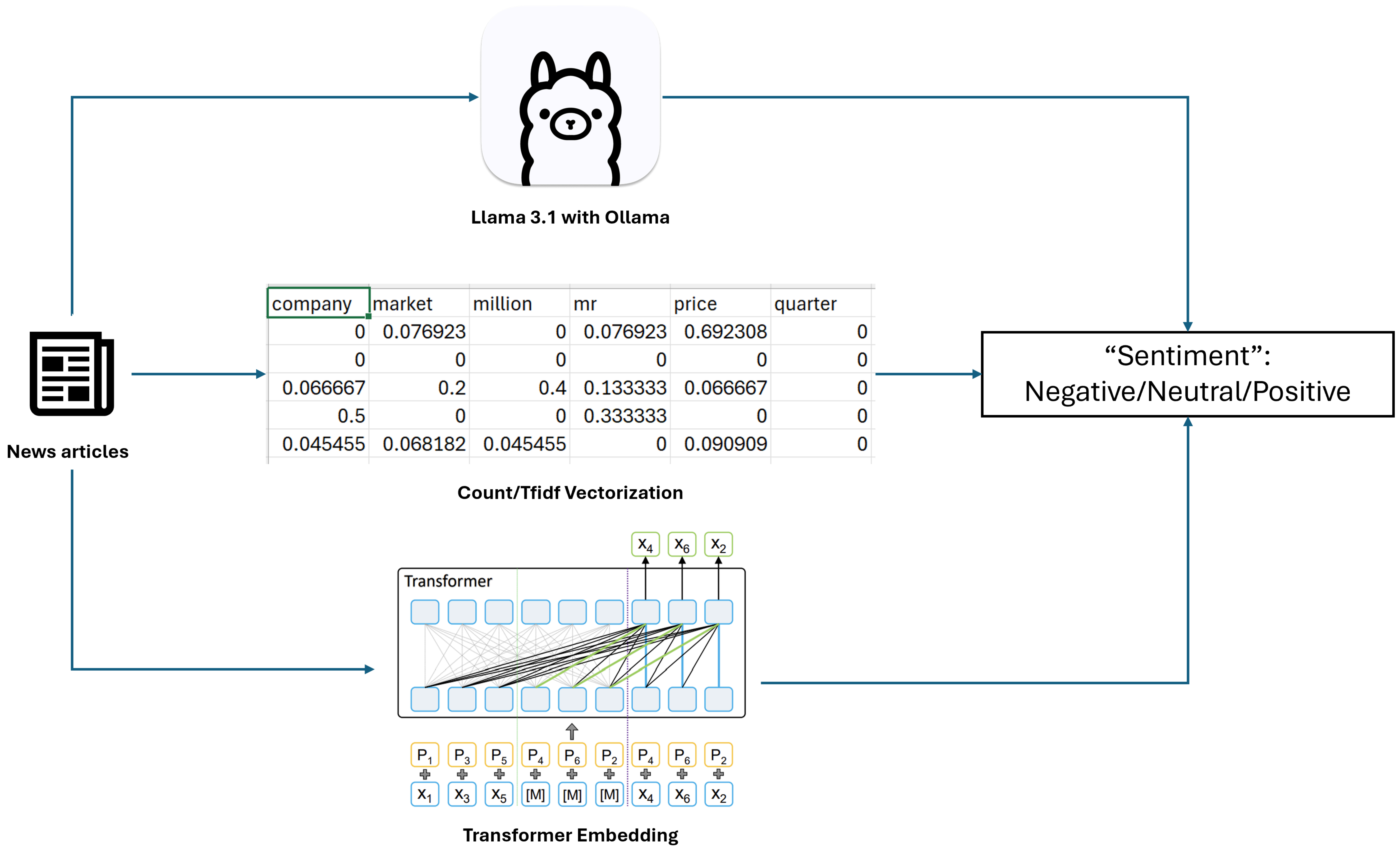
CSC401/2511 Assignment 1 Tutorial 2

Winston (Yuntao) Wu

University of Toronto

Overview


Llama 3.1 with Ollama
News articles
Count/Tfidf Vectorization
"Sentiment": Negative/Neutral/Positive
Transformer
X_4 X_6 X_2
P_1 P_3 P_5 P_4 P_6 P_2 P_4 P_6 P_2
X_1 X_3 X_5 [M] [M] [M] X_4 X_6 X_2
Transformer Embedding

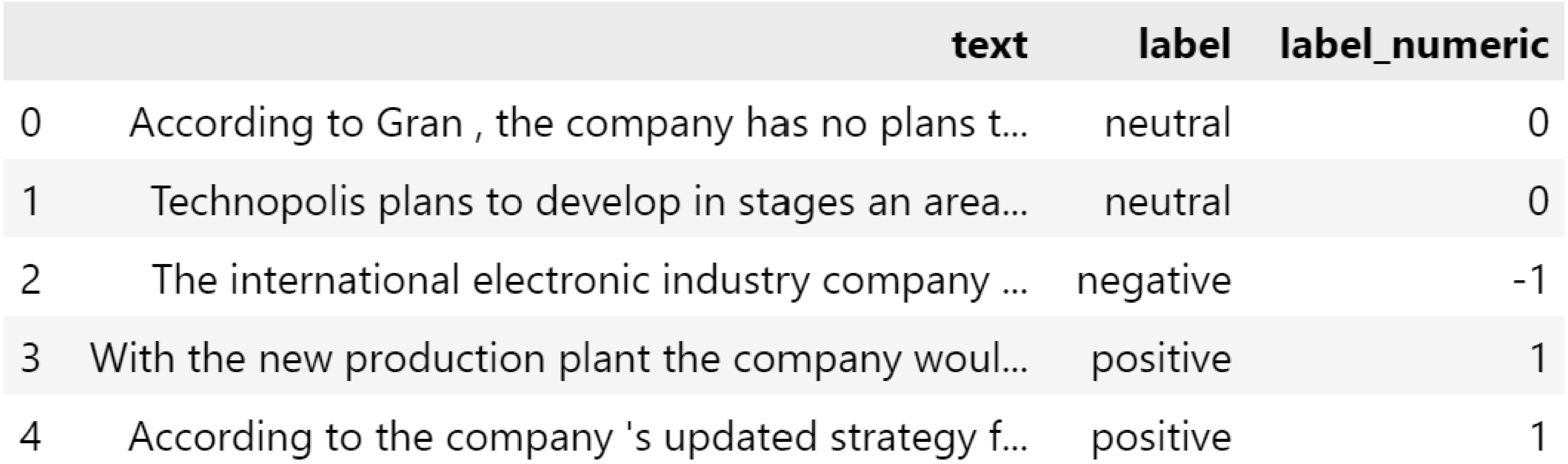
company	market	million	mr	price	quarter
0	0.076923	0	0.076923	0.692308	0
0	0	0	0	0	0
0.066667	0.2	0.4	0.133333	0.066667	0
0.5	0	0	0.333333	0	0
0.045455	0.068182	0.045455	0	0.090909	0

Image Source1 Image Source2

Dataset (FPB)

Financial Phrase Bank:

- Polar sentiment dataset of sentences from financial news
- 4k sentences from English language financial news rated by 5-8 annotators
- The dataset can be accessed from `/u/cs401/A1/data/fpb_dataset.parquet`



	text	label	label_numeric
0	According to Gran , the company has no plans t...	neutral	0
1	Technopolis plans to develop in stages an area...	neutral	0
2	The international electronic industry company ...	negative	-1
3	With the new production plant the company woul...	positive	1
4	According to the company 's updated strategy f...	positive	1

Dataset (WSJ89)

Wall Street Journal (1989):

- 2k articles from 1989 Wall Street Journal from the English Penn Treebank corpus. Dataset is at `/u/cs401/A1/data/wsj.gz`
- We tag each article by the sign of 5-day log return of S&P500 on previous day ($\log \frac{p_{t-1}}{p_{t-6}}$). Tags are save in `/u/cs401/A1/data/wsj89_labels.parquet`

```
wsj
├── 00
│   ├── wsj_0001.gz
│   └── wsj_0002.gz
|   ...
├── 01
│   ├── wsj_0100.gz
│   ├── wsj_0101.gz
│   ...
...
```

`wsj.gz` File Structure

```
.START

After a bad start, Treasury bonds were buoyed
by a late burst of buying to end modestly
higher.

"The market was pretty dull" for most of the
day, said Robert H. Chandross, vice president
at Lloyds Bank PLC.
He said some investors were reluctant to plunge
into the market ahead of several key economic
indicators due this week, especially Friday's
potentially market-moving employment report.
```

Sample article

	fn	label_numeric
0	wsj_0001.gz	-1.0
1	wsj_0002.gz	-1.0
2	wsj_0003.gz	-1.0
3	wsj_0004.gz	1.0
4	wsj_0005.gz	-1.0

`wsj89_labels.parquet`

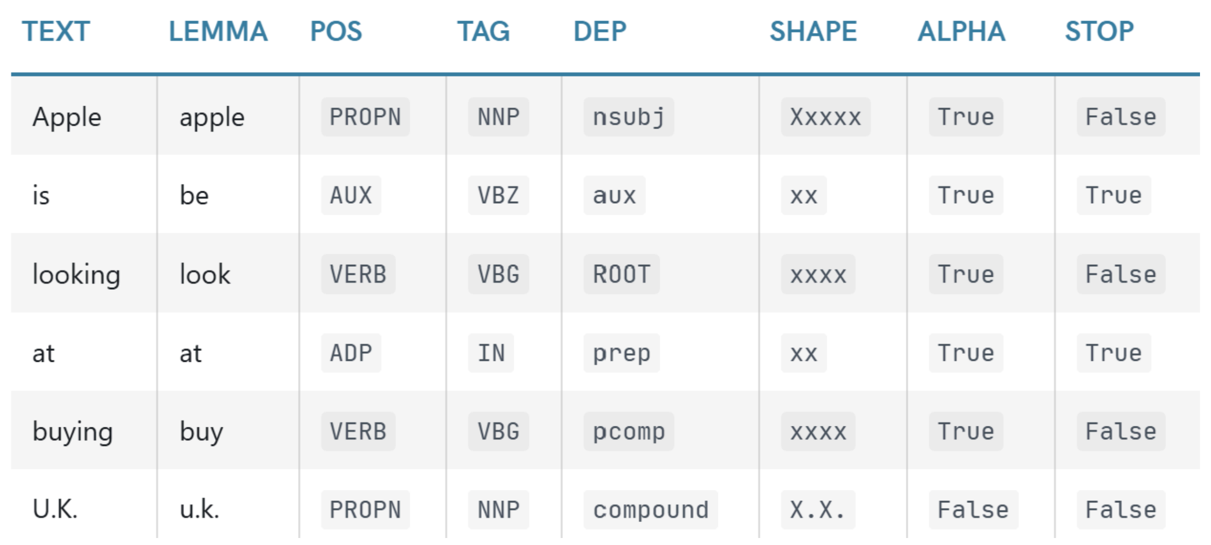
Step 1: Preprocess

Your tasks:

- Process the raw `wsj.gz` file, by properly iterating through all the contained gz files and extract out the text.
- Then merge `wsj89_labels.parquet`, using `pd.merge(df1, df2, on="fn")`.
- Cleanup texts:
  - Replace whitespace characters e.g. `\n, \t`
  - Use `html.unescape` to decode all html characters
  - Remove URLs, numerical values, multiple, leading/tailing white spaces. RegEx (`re.sub`) will be helpful
  - Lemmatize & remove stop words, puntuations, digits using spaCy.

Step 1 Spacy

spaCy is a library for advanced Natural Language Processing in Python and Cython. It can be used for tagging, parsing, named entity recognition, text classification and more. However, we only use its ability to obtain lemmata, along with token type detection.



TEXT	LEMMA	POS	TAG	DEP	SHAPE	ALPHA	STOP
Apple	apple	PROPN	NNP	nsubj	Xxxxx	True	False
is	be	AUX	VBZ	aux	xx	True	True
looking	look	VERB	VBG	ROOT	xxxx	True	False
at	at	ADP	IN	prep	xx	True	True
buying	buy	VERB	VBG	pcomp	xxxx	True	False
U.K.	u.k.	PROPN	NNP	compound	X.X.	False	False

Note: Always take spaCy's output to be correct when completing the tokenization and lemmatization.

lemma: an abstract conceptual form of a word that has been mentally selected for utterance in the early stages of speech production.

- E.g. lemma (best) = good (degree)
- E.g. lemma (houses) = house (number/amount)
- E.g. lemma (housing) = housing

Image Source

Step 1 Running Code and Submission

After you complete the code in `a1_preprocess.py`, you can run the following code on the server to process FPB and WSJ89:

```
python3.10 a1_preprocess.py --filename_prefix wsj89
python3.10 a1_preprocess.py --filename_prefix fpb
```

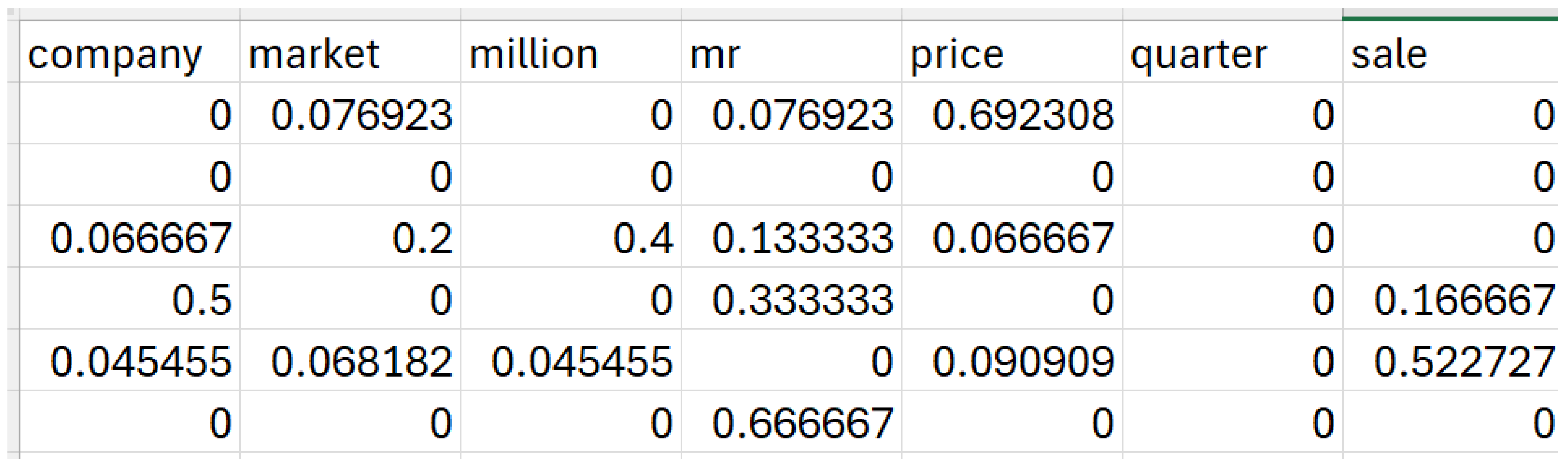
Files to submit:

- `a1_preprocess.py`

Step 2: Vectorize (Statistical)

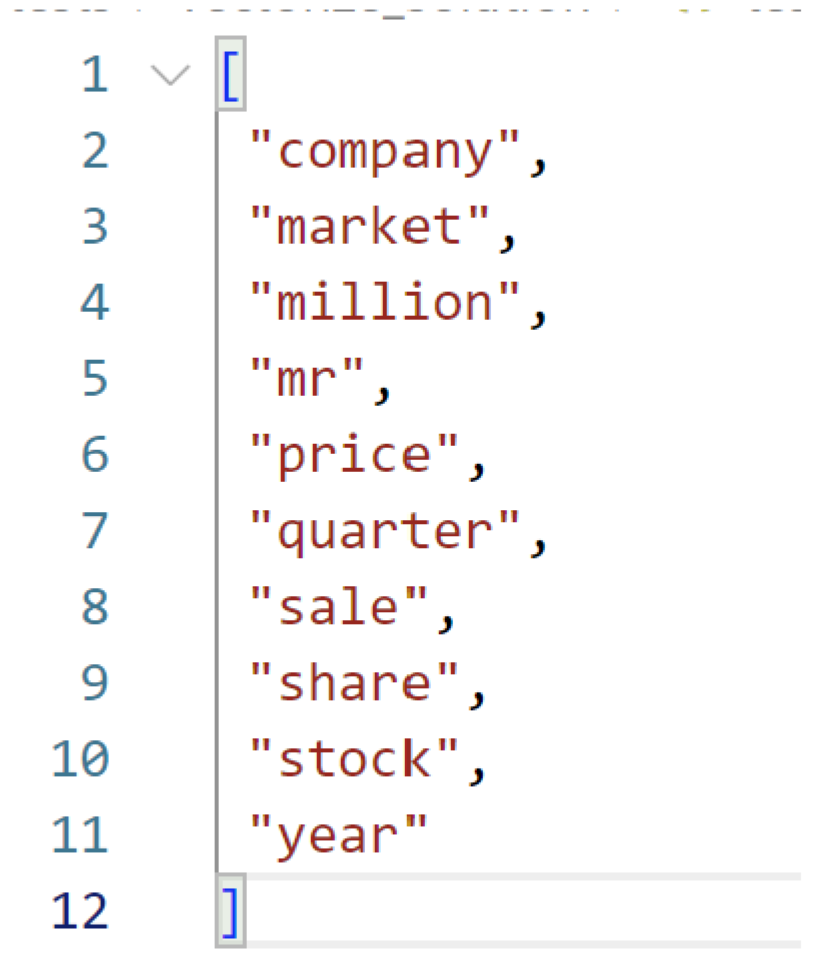
You will work with the following statistical vectorizers to extract features from FPB and WSJ89:

- Count Vectorizer: max_features=10000, min_ngrams=1, max_ngrams=3 (**Note**: You may want to add `1e-8` in the division for stability when computing frequency)
- Tf-idf Vectorizer: max_features=10000, min_ngrams=1, max_ngrams=3



company	market	million	mr	price	quarter	sale
0	0.076923	0	0.076923	0.692308	0	0
0	0	0	0	0	0	0
0.066667	0.2	0.4	0.133333	0.066667	0	0
0.5	0	0	0.333333	0	0	0.166667
0.045455	0.068182	0.045455	0	0.090909	0	0.522727
0	0	0	0.666667	0	0	0

Sample output for vectorized text



```
[
 "company",
 "market",
 "million",
 "mr",
 "price",
 "quarter",
 "sale",
 "share",
 "stock",
 "year"
]
```

Sample output for feature names

Step 2: Vectorize (MPNET)

MPNET is a BERT-based model, which adopts pre-training on masked and permuted language modeling. You will learn more details about transformer models in this course. Here, we use its basic functionality of text embedding with transformers library.

```
# load the model using transformers AutoClass
tokenizer = AutoTokenizer.from_pretrained("sentence-transformers/all-mpnet-base-v2",
cache_dir="/u/cs401/A1/model_weights")
model = AutoModel.from_pretrained("sentence-transformers/all-mpnet-base-v2",
cache_dir="/u/cs401/A1/model_weights")
```

```
# Tokenize text (with truncation) and forward the data
encoded_input = tokenizer(text, padding=True, truncation=True, return_tensors='pt')
model_output = model(**encoded_input)
```

You need to apply **mean_pooling** on the last hidden layer output (`model_output[0]`) to complete the embedding. Only positions with attention_mask=1 should be included in the mean value. You should figure out how to properly change the shapes.

```
# embeddings from model_output has shape (B, L, N)
# Attention masks has shape (B, L)
# B is batch size, L is sequence length, N is feature size
torch.sum(token_embeddings * attention_mask, 1) / torch.clamp(input_mask_expanded.sum(1), min=1e-9)
```

Note: You should run this model on raw text, same as Llama3.1

Note: You can use GPU to run this model.

Note: your vectorized version should have feature size=768, the feature list should be `["feat_0","feat_1",...,"feat_767"]`

Step 2 Running Code and Submission

After you complete the code in `a1_vectorize.py`, you can run the following code on the server to process FPB and WSJ89:

```
python3.10 a1_vectorize.py --filename_prefix wsj89 --vectorizer_type count
python3.10 a1_vectorize.py --filename_prefix wsj89 --vectorizer_type tfidf
srun -p csc401 --gres gpu python3.10 a1_vectorize.py --filename_prefix wsj89 --vectorizer_type mpnet --data_column text
python3.10 a1_vectorize.py --filename_prefix fpb --vectorizer_type count
python3.10 a1_vectorize.py --filename_prefix fpb --vectorizer_type tfidf
srun -p csc401 --gres gpu python3.10 a1_vectorize.py --filename_prefix fpb --vectorizer_type mpnet --data_column text
```

Files to submit:

- `a1_vectorize.py`

Step 3.1: Classification with GaussianNB and MLP

- In this part, you will use the `GaussianNB` and `MLPClassifier` from scikit-learn to perform classification.
  - Use default parameters for `GaussianNB`. It should be fast (within 1 second)
  - To speed up convergence of the `MLPClassifier`, remember to set the `alpha` parameter to 0.05. Count and Tf-idf vectorizers generate a sparse matrix of 10000 features, it will take a long time for MLP to converge on CPU.
- Properly split the data by `train_test_split(self.X, self.y, train_size=train_size, random_state=401)`. The random state is important. You need to have the same split for comparison with Llama3.1. Do not change it.

Step 3.2: Classification with Top Features

For this part, please simply follow the instruction in the handout.

You need to provide the list of most important features, and train **the best model** on these important features.

In addition, you need to name the features extracted by **Count** and **Tf-idf** vectorizers using `feature_names.json`. For example, if you have feature indices: `[64, 128, 256, 512, 1024]`. You should extract out the corresponding n-gram list from `feature_names.json`. The feature names could be `market, eur mn, decrease, profit, disclose`.

Then also briefly comment on why those features may be important.

Note: the important features may be the same for Count and Tf-idf vectorizers. You can explain in one file why they may be important, and mention in the other file that the question has been answered. It will be manually checked by grading TAs.

Example output:

```
5 p-values: [a, b, c, d, e]
50 p-values: [50 values]
Accuracy for full dataset: acc float
Top-5: { 5 indices }
```

Useful Function

```
X_new = SelectKBest(f, k=5).fit_transform(X, y)
X_new.get_support()
```

Step 3.3: Classification with Cross Validation

Generate a 5-Fold split on the entire dataset:

```
kf = KFold(5, shuffle=True)
kf.split(X, y) # this gives [(train_idx1, test_idx1),...,(train_idx5, test_idx5)]
```

Then train both models (`GaussianNB` and `MLPClassifier`) on five folds, get accuracies array on test sets and p -value for independent T-test (to test whether there is a statistically significant difference between the means in two unrelated groups.)

Example output:

```
Fold 0 Accuracies: [model 0 acc, model 1 acc]
Fold 1 Accuracies: [model 0 acc, model 1 acc]
Fold 2 Accuracies: [model 0 acc, model 1 acc]
Fold 3 Accuracies: [model 0 acc, model 1 acc]
Fold 4 Accuracies: [model 0 acc, model 1 acc]
p-values: [p-val]
```

5-fold Cross-Validation

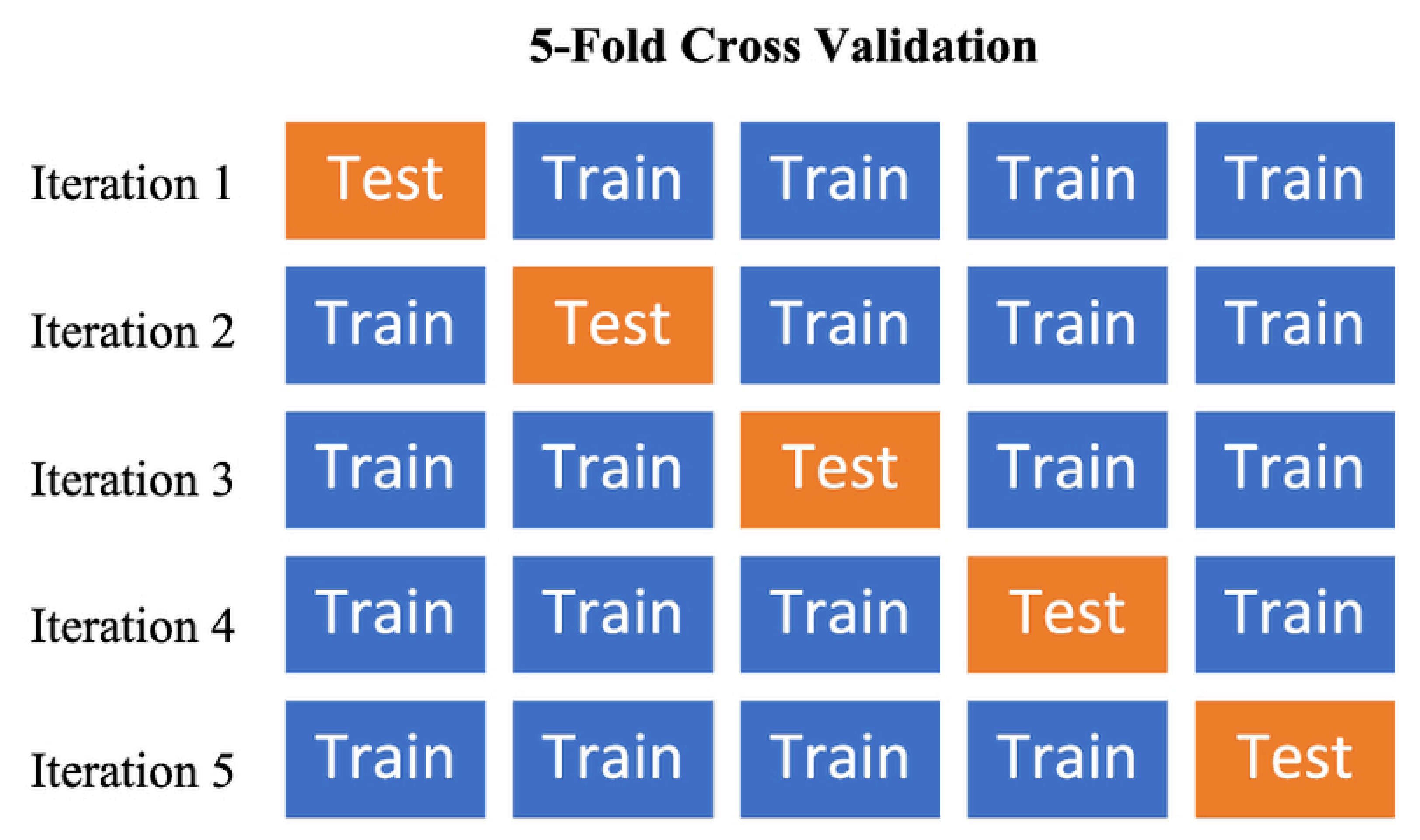


Image Source

Step 3 Running Code and Submission

After you complete the code in `a1_classify.py`, you can run the following code on the server to process FPB and WSJ89:

```
python3.10 a1_classify.py --filename_prefix wsj89 --vectorizer_type count
python3.10 a1_classify.py --filename_prefix wsj89 --vectorizer_type tfidf
python3.10 a1_classify.py --filename_prefix wsj89 --vectorizer_type mpnet

python3.10 a1_classify.py --filename_prefix fpb --vectorizer_type count
python3.10 a1_classify.py --filename_prefix fpb --vectorizer_type tfidf
python3.10 a1_classify.py --filename_prefix fpb --vectorizer_type mpnet
```

Note: This will generate 18 files in total (2 dataset, 3 vectorizers each, 3 tasks for each vectorizer and each dataset)

Files to submit:

- `a1_classify.py`
- All 18 generated files. In `a1_classify_xx_top_feats_{count/tfidf}.txt`, report the top 5 feature names with comments
- `a1_compare.txt`: A brief analysis comparing the performance of Llama 3.1 to traditional vectorization and classification methods.

Note: The current Llama 3.1 on teach.cs server may fail on texts that are too long because of constrained GPU VRAM. You can truncate the text to below 512 tokens and report the results. Document the change.

Notes

- You should only modify code in specific sections where we highlight with ``Your code goes here'' and TODO blocks. No additional libraries should be imported, unless we announce otherwise.
- The performance of llama will vary across different selection of data points. It is possible to have a very low/very high accuracy on 25 samples. What we look for is your analysis on how it performs and what the potential reasons are.
- You do not need to worry about the following warnings from python or any warnings filtered in the code:
  - Torch User Warning: Can't initialize NVML, and any warnings associated with CUDA when running on CPU machines
  - Pandas Future Warning: Setting an item of incompatible dtype is deprecated
- The provided runtime statistics are only an estimation. The actual runtime of the programs depends on your implementation, the server load, and many other factors.
- You should not have to download any models. Make sure you properly set the `cache_dir` for mpnet, and use the correct python version (`python3.10`).
- When accuracy/precision/recall/frequency cannot be properly calculated due to zero denominator, default output to 0.