Computer Science 401 6 January 2026
St. George Campus University of Toronto

Homework Assignment #1
Due: Thursday, 5 Februrary 2026 at 11:59pm

Financial Sentiment Analysis

TAs: Winston (Yuntao) Wu. Email: csc401-2026-01-a1@cs.toronto.edu.

Overview

In this assignment, you will develop classifiers to predict the sentiment of financial texts, focusing on both intrinsic sentiment and its effect on stock price movement. This assignment will provide you with experience using the Financial Phrase Bank and Wall Street Journal corpora, Python programming for natural language data, sentiment analysis, machine learning with scikit-learn, and Llama 3.1.

First, you will evaluate the performance of Llama 3.1 on a sentiment analysis task using the Financial Phrase Bank. Then, you will build a new feature-based classifier from scratch. This process includes pre-processing the corpus, designing and implementing feature extractors to gather information from each article, and creating classifiers to predict historical stock movements based on textual features using various machine learning algorithms. Finally, you will compare the results of your classifiers with a zero-shot large language model.

Please monitor the course bulletin board (Piazza) regularly for announcements and discussion pertaining to this assignment.

Dataset Format

In this assignment, you will work with the parquet file format. The Pandas library will be extensively used for reading and writing parquet files.

Financial Phrase Bank

The Financial Phrase Bank (FPB) is a three-valued sentiment dataset comprising sentences from financial news, available on (HuggingFace). The dataset contains sentences from English-language financial news, categorized by sentiment. It is divided based on the agreement rate of 5-8 annotators, and we use instances with at least 50% annotator agreement. The dataset can be retrieved from `/u/cs401/A1/data/fpb_dataset.parquet`. It includes three columns: `text` (raw news sentences), `label` (string label of sentiment: positive, neutral, or negative), `label_numeric` (numerical label of sentiment: 1 for positive, 0 for neutral, and -1 for negative).

Wall Street Journal

We collected articles from the 1989 Wall Street Journal (WSJ) from the English Penn Treebank 2 (PTB-2) corpus. Typically, news articles report events that occurred the previous day. We tag all news articles on day t as positive if the 5-day log return on the previous day ($\log \frac{p_{t-1}}{p_{t-6}}$) is positive, and as negative if

Copyright © 2024-2026, Ken Shi, Winston Wu and Gerald Penn. All rights reserved.

the 5-day log return on the previous day is negative. Your goal with the WSJ dataset is to use the news text to predict whether the market performs well (positive return) or not (negative return). Note that the 5-day log return is the only indicator chosen for prediction in this assignment, and you are only tasked with predicting the provided indicator. The text dataset should be used in situ `/u/cs401/A1/data/wsj89`, and the labels can be retrieved from `/u/cs401/A1/data/wsj89_labels.parquet`. The parquet file includes two columns: `fn` (The original filename of the news article in the PTB-2), and `label_numeric` (1 if 5-day log return is positive, -1 if it is negative. NaN if invalid).

Starter Code

We provide five starter code files: `a1_utils.py`, `a1_preprocess.py`, `a1_vectorize.py`, `a1_classify.py` and `a1_llama3.py`. A `sanity_check.py` file is also provided for you to partially validate your work. Please read the instructions in both the starter code and this handout carefully.

Part 1

1 Llama 3.1 Analysis [10 marks]

The first part of this assignment is designed to help you become familiar with Llama 3.1 for sentiment analysis and to get started with basic accuracy calculations for classification tasks.

1.1 Evaluation Utils

In `a1_utils.py`, write the code to compute the following manually, using the associated function templates:
Accuracy : The total number of correctly classified instances over all classifications: $A = \frac{\sum_i c_{i,i}}{\sum_{i,j} c_{i,j}}$.
Recall : For each class κ, the fraction of cases that are truly class κ that were classified as κ, $R(\kappa) = \frac{c_{\kappa,\kappa}}{\sum_j c_{\kappa,j}}$.
Precision : For each class κ, the fraction of cases classified as κ that truly are κ, $P(\kappa) = \frac{c_{\kappa,\kappa}}{\sum_i c_{i,\kappa}}$.

1.2 Querying Llama 3.1

We have deployed a Llama 3.1 8B Instruct model on the `teach.cs` server. The model is loaded with 4-bit quantization using Ollama. Your query should be based on the default settings:

```
max_new_tokens=256,
temperature=0.5,
top_p=0.95,
top_k=10,
```

Functionality In `a1_llama3.py`, the code to submit a process request to the server is provided. The `send_request` function takes your UTORid and raw text string as inputs and returns a JSON object from Ollama. You need to complete the `parse_response` function to generate a JSON object in the following form:

```
{
    "label": extracted text label (POSTIVE, NEUTRAL, NEGATIVE) from the response,
    "label_numeric": 1/0/-1 for positive, neutral, negative,
    "raw_result": res["message"]["content"],
    "compute_time": total_duration in second
}
```

Your Tasks In this part, you will sample 25 articles from `/u/cs401/A1/data/fpb_dataset.parquet`, and compute the sentiment scores using Llama 3.1 with `send_request`. Obtain the 3×3 confusion matrix C using `confusion_matrix`. $c_{i,j}$, the element at row i, column j in C, is the number of instances belonging to class i that were classified as class j. Then compute the accuracy, precision, and recall rates. There are two files you need to modify:

1. In `a1_utils.py`, implement the functions to compute accuracy, precision and recall. NumPy is the only library allowed.
2. In `a1_llama3.py`, replace the lines marked with `TODO` with the code to read the text from the Financial Phrase Bank (FPB) and compute sentiment using the Llama 3.1 endpoint. Then, calculate the accuracy, precision, and recall rates, and store the results in `a1_part1.txt` using the provided format string.
3. In `a1_part1.txt`, comment on the performance of Llama3.1, based on the accuracy, precision, recall rate, and samples of the generated explanation (at least one for correct and at least one for incorrect, with your comments).

Note

1. Computing sentiment scores using Llama 3.1 takes around 4 seconds per article. The server operates on a queuing system, so you may experience delays. Cache the results in a dataframe and save them locally to avoid repeated endpoint calls.
2. If it is not giving any of POSITIVE, NEUTRAL, NEGATIVE, you assume it is NEUTRAL. You can modify system prompt and parameters, but you need to document the change and metrics before and after the change. Changing the prompt is *not a required component* of this assignment, and you will not be graded on how well your prompt works.
3. Truncate the text and increase the delay if the requests are constantly failing.
4. *Only one Llama 3.1 server is available for both CSC 401/2511 and CSC 485/2501.* Start early to avoid long wait times in the request queue.
5. Use the provided server exclusively for assignments, not personal projects. Each UTORid will be rate-limited.
6. Use your own UTORid in your requests. Only requests from registered UTORids will be processed. We will track the total number of requests per UTORid as partial validation of your work.

Part 2

2 Step 1: Pre-processing [18 marks]

The FPB dataset is available in a pandas dataframe, but the WSJ dataset is in a raw gz file. You need to properly extract the text from the gz file, and merge the labels, following the instructions given in `a1_preprocess.py`.

The FPB and WSJ articles, as given, are not in a form amenable to feature extraction for classification – there is too much "noise". Therefore, the first step is to complete the class name, `A1Preprocess`, in accordance with General Specifications.

1. Replace all non-space whitespace characters, including newlines, tabs, and carriage returns, with spaces.
2. Replace HTML character codes (i.e., *&...;*) with their ASCII equivalents (see `http://www.asciitable.com`).
3. Remove all URLs (i.e., tokens beginning with *http* or *www*).
4. Remove all numerical values (0-9).
5. Remove duplicate spaces between tokens and leading/trailing spaces.
6. Convert the text to lower case, and apply the following steps using spaCy (see below):

- Replace the token itself with the `token.lemma_`.
- Remove digit, stop words and punctuation as classified by spaCy.

Additional Specifications

spaCy The package spaCy is very handy for many NLP tasks. Here, we **only** use its ability to obtain lemmata, along with token type detection. For example:

```
import spacy

nlp = spacy.load("en_core_web_sm", disable=["parser", "ner"])
nlp.add_pipe("sentencizer")

doc = nlp("I know words. I have the best words")

for token in doc:
    print(token.text, token.lemma_, token.is_stop, token.is_punct, token.is_digit)
```

Functionality The `A1Preprocess` class in `a1_preprocess.py` file reads a pandas dataframe, processes each value in the "text" column, and saves the processed text in the "cleaned_text" column. The processed file will be saved to your local output directory for usage in the following tasks.

The program takes three arguments: output_dir (optional; default="./output"), a1_dir (optional; default="/u/cs401/A1"), and filename_prefix (optional; default="wsj89").

Your Tasks There are three parts you need to modify in `a1_preprocess.py`:

1. In `generate_wsj_dataframe`, replace the lines marked with `TODO` with the code they describe.
2. In `A1Preprocess`, replace the lines marked with `TODO` with the code they describe.
3. In the main block, generate the WSJ dataframe, read the dataframe from the parquet file with the provided prefix, process the texts in the entire dataframe, and save the output to your local output path.
4. Run the following commands to generate the dataframe, for both FPB and WSJ.

```
python3.12 a1_preprocess.py --filename_prefix wsj89
python3.12 a1_preprocess.py --filename_prefix fpb
```

For this section, you may only use standard Python libraries, pandas and spaCy. For debugging, you are advised to either use a different input folder with your own dataframe, stored as a csv or parquet.

3 Step 2: Vectorization [22 marks]

The second step is to complete a Python program named `a1_vectorize.py`, in accordance with General Specifications. This program will take the preprocessed dataframes from Step 1 and extract features using the following three vectorizers for training models and classifying texts in Step 3:

- **Count Vectorizer**: Use `CountVectorizer` from scikit-learn to extract the top 1,000 unigrams, bigrams, and trigrams with the default parameters: `max_features=1,000`, `min_ngrams=1`, `max_ngrams=3`
- **Tf-idf Vectorizer**: Use `TfidfVectorizer` from scikit-learn, extract the top 1,000 unigram, bigram and trigrams, with the default parameters provided in the file: `max_features=1,000`, `min_ngrams=1`, `max_ngrams=3`
- **MPNET Embedding**: Load the `sentence-transformers/all-mpnet-base-v2` model using the transformers library, compute token embeddings, perform mean pooling, and normalize the results. The model is cached at `'/u/cs401/A1/model_weights'`; use this path as `cache_dir` when loading the model.

  Use `padding=True, truncation=True, return_tensors='pt'` for tokenization. Truncate the text input if it exceeds the acceptable length for MPNET. Check HuggingFace for more specifications.

The output of `a1_vectorize.py` will be used in Step 3.

Your Tasks You need to modify the following methods in the `A1Vectorize` class in `a1_vectorize.py`:

1. Initialize the vectorizers corresponding to "vectorizer_type" in `__init__`.
2. Implement `mean_pooling` as instructed. Note that it is only used for MPNET.
3. Implement `vectorize` as instructed. For the count vectorizer, normalize the count into frequency by the total count of vocabularies.

Build the vectorized dataframe and feature name list by calling "run" in the main block. The following commands processes the WSJ dataset. Change filename_prefix in order to run on the FPB. For MPNET, run the vectorizer on the *raw text*. For the count and tf-idf vectorizers, run the vectorizer on *cleaned text*. You should not need to change other arguments.

```
python3.12 a1_vectorize.py --filename_prefix wsj89 --vectorizer_type count
python3.12 a1_vectorize.py --filename_prefix wsj89 --vectorizer_type tfidf
srun -p csc401 --gres gpu python3.12 a1_vectorize.py --filename_prefix wsj89 \
--vectorizer_type mpnet --data_column text
```

You can vectorize with MPNET using the CPU, which takes $\sim$ 1 minute for WSJ and FPB.

4 Step 3: Experiments and Classification [30 marks]

The third step is to use the features extracted in Step 2 to classify articles using the scikit-learn machine learning package. You will modify various hyperparameters and interpret the results analytically, discussing your findings with scientific rigour. Complete the `main` body and the `A1Classify` class in `a1_classify.py` for the specified experiments.

The program takes four arguments, but you only need to focus on two: filename_prefix (to choose the dataset, WSJ or FPB), and vectorizer_type (to select the vectorization output from the previous step). The following commands run classification on WSJ with the Tf-idf vectorizer and on FPB with MPNET, respectively.

```
python3.12 a1_classify.py --filename_prefix wsj89 --vectorizer_type tfidf
python3.12 a1_classify.py --filename_prefix fpb --vectorizer_type mpnet
```

You should create the output directory if it doesn't already exist. In `main`, you are expected to load the dataframe and feature-names list, initialize a `A1Classify` class with appropriate input parameters, and call the experimental functions in order. Use the `train_test_split` method to split the data into a random 80% for training and 20% for testing in the first two experiments.

4.1 Classifiers

Train the following classifiers (see hyperlinks for API) with `fit(X_train, y_train)`:

1. `GaussianNB`: a Gaussian naive Bayes classifier.
2. `MLPClassifier`: A feed-forward neural network, with $\alpha = 0.05$.

Here, `X_train` is the vectorization of your training data, and `y_train` is the vectorization of the target variable. Obtain predicted labels with these classifiers using `predict(X_test)`, where `X_test` is the vectorization of your testing data. Obtain the $n \times n$ confusion matrix C using `confusion_matrix`, where n is the number of classes ($n = 2$ for WSJ, and $n = 3$ for FPB). $c_{i,j}$, the element at row i, column j in C, is the number of instances belonging to class i that were classified as class j. Compute the accuracy, recall and precision manually, using the evaluation metrics defined in 1.1.

Write the results to the text file `a1_classify_{filename_prefix}_{vectorizer_type}.txt` in the output directory. You must write to this file using the format strings provided in the template. **If you do not follow the format, you may receive a mark of zero**. For each classifier, you will print the accuracy, recall, precision, and confusion matrix. You may include a written analysis if you are so inclined, but only after the results.

4.2 Feature Analysis

Certain features may be more or less useful for classification, and too many can lead to overfitting or other problems. Here, you will select the best features for classification using `SelectKBest` according to the f_classif metric as in:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

selector = SelectKBest(f_classif, you_figure_it_out)
X_new = selector.fit_transform(X_train, y_train)
pp = selector.pvalues_
```

In the example above, `pp` stores the p-value associated with doing a χ^2 statistical test on each feature. A smaller value means the associated feature better separates the classes. Do this:

1. For the training set and each $k \in \{5, 50\}$, find the best k features according to this approach. Write the associated p-values to `a1_classify_{filename_prefix}_top_feats_{vectorizer_type}.txt` using the format strings provided.
2. Train the *best* classifier from section 4.1 on the training set, using only the best $k = 5$ features. Write the accuracy to the same file using the format strings provided.
3. Extract the indices of the top $k = 5$ features.
4. Following the above, answer the following questions for count and tfidf vectorization:
   (a) Name the top 5 features chosen using feature names extracted in Step 2.
   (b) Hypothesize as to why those particular features (n-grams) might differentiate the classes.

4.3 Cross-validation

Many papers in machine learning stick with a single subset of data for training and another for testing (occasionally with a third for validation). This may not be the most honest approach. Is the best classifier from Section 4.1 *really* the best? For each of the classifiers in Section 4.1, run 5-fold cross-validation given all the initially available data. Specifically, use `KFold`. **Set the shuffle argument to true**.

For each fold, obtain the accuracy on the test partition after training on the rest for each classifier. Report the accuracy of each classifier for each of the 5 folds in the order specified in 4.1 to `a1_classify_{filename_prefix}_cross_valid_{vectorizer_type}.txt` using the format strings provided. Next, determine whether the accuracy of your best classifier, across the 5 folds, is *significantly* better than any of the others. That is, given vectors `a` and `b`, one for each classifier, containing the accuracy values for each of the respective 5 folds, obtain the p-value from the output `S`, below:

```
from scipy import stats
S = stats.ttest_ind(a, b)
print(S.pvalue)
```

You should have one p-value. Report it using the provided format string, excluding the self-comparison.

Your Tasks You need to modify the following methods in the `A1Classify` class in `a1_classify.py`:

1. Implement `classify`, `classify_top_feats`, and `classify_cross_validation` by following the provided instructions.
2. Run the experiments on both the WSJ and FPB datasets, with all of the count, tf-idf and MPNET vectorizations. What follows are two of the commands:

```
python3.12 a1_classify.py --filename_prefix wsj89 --vectorizer_type tfidf
python3.12 a1_classify.py --filename_prefix fpb --vectorizer_type mpnet
```

On WSJ, the MLP-based classifier can take around 100 seconds to run on one pass, and 10 minutes 5-folded. On FPB, it takes around 6 minutes to run on one pass, and 20 minutes 5-folded.

Comparing with Llama 3.1 Now modify the `a1_llama3.py` file (main block and the prompt). Obtain the WSJ test data from the classification task and select a subsample of 25 data points using your UTORid as the random seed. Compute the sentiment using Llama 3.1 on teach.cs, then calculate the accuracy, precision, and recall. Record the indices of the selected articles and compute the accuracy, precision, and recall using the classifier you built in Step 3, utilizing the best vectorization method and classification model from Step 3. In `a1_compare.txt`, record the following:

1. The indices of the 25 randomly sampled articles.
2. Accuracy, precision, recall rates of the Llama 3.1 classification.
3. Accuracy, precision, and recall rates of the best vectorization and classification model.

4. If applicable, a brief discussion on how you modified the prompt to enable Llama 3.1 to produce two labels (positive/negative) for the WSJ dataset and three labels (positive/neutral/negative) for the FPB dataset. You will *not* be penalized if you choose not to modify the prompt and instead generate three labels for both WSJ and FPB.
5. A brief analysis comparing the performance of Llama 3.1 to the other vectorization and classification methods. Consider the following questions:
   (a) Which method has higher accuracy?
   (b) What "sentiment" does each method capture? (**Note**: It is not simply because Llama is providing three labels while statistical method is providing two labels.)
   (c) Why does the method perform better?

Note: The availability of the Llama 3.1 endpoint at particular times is not guaranteed, especially when it gets close to the deadline. Start early!

General Specifications

As part of grading your assignment, the grader may run your programs and/or Python files on test data and configurations that you have not previously seen. This may be done in part by automatic scripts. It is therefore important that each of your programs precisely meets all the specifications, including its own name and the names of the files and functions that it uses. **A program that cannot be evaluated because it varies from specifications will receive zero marks on the relevant sections.**

The submitted files should be generated on the `teach.cs` machines. Do **not** hardwire the absolute address of your home directory within the program (except for your `__main__` block, which won't be run in autotests); the grader does not have access to this directory.

All your programs must contain adequate internal documentation to be clear to the graders.

We use Python version 3.12.13 on teach.cs.

Submission Requirements

This assignment is submitted electronically. You should submit:

1. All your code for `a1_utils.py`, `a1_preprocess.py`, `a1_vectorize.py`, `a1_classify.py` and `a1_llama3.py`.
2. `a1_part1.txt`: Report the initial results for Llama 3.1 on the FPB.
3. 18 files for Section 4, with names properly formatted as specified. Each dataset should have 9 files: 3 files for each of the 3 tasks in Section 4 for a specific vectorization.
4. `a1_compare.txt`: Comparison between the two types of classifiers: feature-based and Llama 3.1.
5. `ID.txt`: In another file called `ID.txt` (use the template on the course web page), provide the following information:
   (a) Your first and last name.
   (b) Your student number.
   (c) Your teach.cs login ID.
   (d) Your preferred contact email address.
   (e) Whether you are an undergraduate or graduate.
   (f) This statement:

   > By submitting this file, I declare that my electronic submission is my own work, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters and the Code of Student Conduct, as well as the collaboration policies of this course.

You do not need to hand in any files other than those specified above. The electronic submission must be made on MarkUS.

Working Outside the Lab

If you want to do some or all of this assignment on your laptop or home computer, you will have to do the extra work of downloading and installing the requisite software and data. You are not allowed to download our copy of the WSJ89 corpus to a non-university machine, however. If you do use your own machine for coding, you take on all of the associated risks. You are strongly advised to upload regular backups of your work to `teach.cs`, so that if your home machine fails or proves to be inadequate, you can immediately continue working on the assignment at `teach.cs`. When you have completed the assignment, you should try all components of your programs on `teach.cs` to make sure that they run correctly there. **Any component that does not work on teach.cs will get zero marks.**

Answers to Frequently Asked Questions

In this section, we provide a list of clarifications to possible concerns. In addition to checking this, students needing clarification with respect to the assignment should check Piazza, where we will have another FAQ section based on questions from your fellow students.

- Always take spaCy's output to be correct when completing the tokenization and lemmatization.
- To speed up convergence of the `MLPClassifier`, remember to set the *alpha* parameter to 0.05. The count and Tf-idf vectorizers generate a sparse matrix of 1000 features, it will take a long time for MLP to converge on a CPU.
- Accuracies in the classification portion may differ from student to student based on a number of benign factors.
- You should only modify code in specific sections that we highlight with "Your code goes here" and TODO blocks. No additional libraries should be imported, unless we announce otherwise.
- You should enclose all your own test codes in the `__main__` block, or remove them before submission, otherwise your tests may time out.
- If you decide to run the code on your own computer, remember to change the program argument to the original version for submission, otherwise your tests will fail.
- The performance of Llama will vary across different selections of data points. It is possible to have a very low/very high accuracy on 25 samples. What we look for is your analysis of how it performs and what the potential reasons are.
- If your Llama 3.1 requests constantly fail, firstly try to increase the initial wait time `time.sleep(2)` to a longer time. If it does not help, contact the teaching assistants on Piazza.
- You do not need to worry about the following warnings from Python:
  - *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 estimates. 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, (`python3.12`).