# 46-924, Fall 2025: Homework #4

Due 11:59 PM (Eastern Time), Tuesday, Dec 2, 2025.

**Homework should be submitted electronically on Gradescope before the deadline. Please submit your homework as a single PDF. We recommend using jupyter notebooks to prepare your homework. Your submission should include all of your code, unless stated otherwise.**

**Be aware that Gradescope requires you to label the locations of the different problems in your pdf, so budget a few minutes for that.**

**Do all computational problems in python. Please post any questions on the Piazza discussion board.**

We are once again (and for the final time!) working with the Financial Phrase Bank dataset 'Sentences_50Agree.txt', which was used in training FinBERT. The Financial Phrase Bank dataset consists of a collection of sentences which were annotated by sixteen people with finance backgrounds. They were asked to label each sentence as either positive, negative, or neutral from the viewpoint of an investor evaluating the impact of a news statement on the stock price. If a sentence is considered to not be relevant from an economic or finanicial perspective then they were supposed to mark it as neutral. Similar to HW1, we are only considering the 'Sentences_50Agree.txt' dataset.

All of the problems in this homework will focus on this multinomial classification problem of labeling the 'financial sentiment' of text as either positive, negative, or neutral. The models will scale up in complexity as you progress through the homework, ending with using your favorite AI tool for labeling the data.

1. *Prep.* You'll start by preparing the dataset that you'll throughout the homework.

   (a) Read in the 'Sentences_50Agree.txt' data and perform the following pre-processing steps:
   - Convert words to lowercase
   - Remove common stop words
   - Remove all numbers
   - Only consider documents more at least 1 word

   Do NOT perform any type of stemming or lemmatization.

   (b) Split your data into training and test data using 'train_test_split' with 90 percent of data for training and 10 percent for testing. To ensure that the training and

test data have the same class proportions, set the 'stratify' argument to be the response variable column . For replicability purposes, set 'random_state = 2013'.

(c) The point of this homework is to resemble the use of a classifier for labeling collections of text. So we will focus on the classification accuracy of all of your models in this homework, treating each type of error equally. To start, report the training data's base rate multinomial classification accuracy on the test data.

2. *Naive Bayes.* Beyond considering base rates, as we discussed in lecture, Naive Bayes classifiers are simple and computationally efficient ways for building text classifiers.

   (a) As your input, compute the TF-IDF values for every word in your pre-processed text for both the training and test data. Make sure you avoid any data leakage between the test and training data when computing the TF-IDF values. *You can feel free to use the sklearn version of TF-IDF for ease.*

   (b) Fit the Multinomial Naive Bayes estimator on the training data with Laplace smoothing, then report the accuracy on the test dataset.

   (c) In a few sentences, describe how the Multinomial Naive Bayes classifier performs against base rates. What does your comparison imply about the relationship between the predictor variables with the response?

3. *Feedforward neural network* Next, we'll compare the performance with a feed-forward neural network.

   (a) Using either MLPClassifier or PyTorch, fit a feedforward neural network that takes the TF-IDF values from the datasets constructed in Problem 2 to predict financial sentiment. For this homework, I'm not requiring you to implement cross-validation for tuning the various neural network hyperparameters. Instead, pick what you believe to be are reasonable choices given this particular **and (in a few sentences) justify your considered neural network hyperparameters.**

   (b) Report the test accuracy of your feed-forward neural network.

   (c) In a few sentences, describe how your feed-forward neural network classifier compares to the Naive Bayes and base rate test data performance. What does your comparison imply about the relationship between the predictor variables with the response?

4. *CNN for text.* You're now going to practice what we discussed in lecture regarding the use of CNNs for text classification in PyTorch.

(a) We'll start with the annoying part, and that's setting up the data for 'torch'. You'll need to do the following to the same train/test split of data defined earlier:

- Tokenize your text data such that it is a list of tokens, rather than a single continuous string.
- Define a vocabulary based on the training data, including tokens for padding '<PAD>' (reserve for 0 index) and unknown tokens in the test data '<UNK>' (reserve for index 1). There are a number of ways to do this, such as the 'vocab' class from torchtext. **Print the size of the vocabulary after you've initialized it.**
- Convert the sentiment labels to integers in order for them to be compatible with torch. You may find LabelEncoder useful here.
- Next (the really annoying part), you need to setup the data loader and iterator. You'll need to create a custom dataset class for this, similar to this PyTorch tutorial. Your custom dataset class should handle the unknown tokens, and then your collate function should handle padding.

Once you have this set-up, initialize the DataLoader object with a batch size of 64 for both the training and test data.

(b) Next, initialize a CNN network with the following architecture:

- Start with an embedding layer via Embedding of learnable weights.
- Next, a convolution layer that takes in the number of embedding dimensions as the number of input channels and some number of filters as the number of output channels, with the kernel size corresponding to the considered n-grams. This convolution layer should generate the same number of filters for multiple choices of kernel sizes (ie searching over 2-grams vs 3-grams vs 4-grams, etc.). Note that in your forward step portion of the code, you'll need to permute the embedding layer to match the dimensionality of the convolution layer.
- For each considered filter size, apply max-pooling over the length of sequence for each of the convolution filters. Then concatenate these together to form the input for a final fully connected layer.
- Use a fully-connected layer to output for the desired number of classes.

We're being lazy here and will not tune these appropriately via cross-validation, but pick values to use for the embedding dimension, number of filters, and considered number of n-grams that you think are appropriate. **Write a few sentences explaining your choices of these values, and report the total number of learnable parameters in your CNN.**

(c) Train your CNN using an appropriate loss function for this problem over a reasonable number of epochs (eg 10). Then report the performance of the CNN on the test dataset.

(d) In a few sentences, describe how your CNN compares to the previous approaches? What does your comparison imply about the relationship between the structure of the input text with the response?

5. *Sequence modeling.* For comparison, you're now going to implement a sequence-based approach for modeling the text sentiment. Fortunately, your dataloader from the previous problem (if done correctly) can be re-used for this problem.

   (a) Next, initialize a version of a sequential model (RNN, LSTM, or GRU) that you think would be appropriate for this problem that meets the following requirements:

   - Start with an embedding layer via Embedding of learnable weights.
   - Next, use some version of a sequential layer with a specified number of hidden units and choose whether or not it is bidirectional.
   - Decide whether or not to include another fully connected layer followed by a non-linear activation function before...
   - Ending with a fully-connected layer to output for the desired number of classes.

   As in the previous problem, we're being lazy here and will not tune these appropriately via cross-validation. But you should use the same number for the embedding dimension as considered for the CNN, and then pick choices for the following: type of sequential layer (RNN, LSTM, GRU), number of hidden units in sequential layer, whether sequential layer is bidirectional, and whether or not you used another fully connected layer before the output. **Write a few sentences explaining your choices of these settings, and report the total number of learnable parameters in your sequential model.**

   (b) Train your sequential model using an appropriate loss function for this problem over a reasonable number of epochs (eg 10). Then report the performance of it on the test dataset.

   (c) In a short paragraph, describe how your sequential model compares to the previous approaches? Are you surprised by the performance of it compared to the CNN? Why or why not? What does your comparison imply about the relationship between the structure of the input text with the response?

6. *Fine-tuning transformers.* You're now going to have the fun of fine-tuning a pre-trained version of BERT for this task using the transformers library that allows you to access models from the Hugging Face repository.

(a) To start, you'll need to re-initialize a version of your dataset such that you are working with the raw text (along with the encoded version of the response variable). Use the same train/test split from before based on setting it to be the same random seed from beginning of this homework. We need to do this because each of the models has their own respective tokenizers and pre-processing steps (which makes your life easier from the perspective of not having to decide for yourself what to consider.)

(b) Now the fun part - fine-tune a light-weight version of BERT known as DistilBERT (we'll discuss the details of this in lecture) for your sentiment classification task on the training data. The version you'll use specifically is 'distilbert-base-uncased'. Update all of the model weights over a small number of epochs (eg 5). **This will take around 30 minutes to run locally**. There are several different ways to implement this fine-tuning procedure, so I'm not going to give you explicit steps (quite frankly, your favorite AI tool will give you decent code for this task...). The main Hugging Face tutorial on fine-tuning is helpful (but you can ignore the step about logging in!), and so is this blog post that is linked from the main DistilBERT page. The steps are somewhat similar to how you set-up the data for the CNN, except the tokenization is handled by the considered model.

(c) Report the accuracy of your fine-tuned version of DistilBERT on the test dataset.

(d) In a few sentences, describe how your fine-tuned version of DistilBERT compares to all of the the previous approaches? What do you think the meaning of this difference is?

7. *Zero-shot AI.* As a final approach for classifying the text, I want you to blindly rely on "AI" to label your test dataset! Include the following in your submission:

   - What "AI" tool did you use (eg ChatGPT, Claude, Gemini)?
   - Copy and paste in your prompt
   - Include your code for reading in the "AI" labeled dataset and report its accuracy on the test dataset.

   In a few sentence: how does this approach compare to all of the previous approaches in this homework? What do you think explains the reasoning for the observed performance of this off-the-shelf classification?