**How I approached the Problem:**

**Part 1: Supervised Learning**

* First off, it was a straightforward task for me since I’m already comfortable with Logistic Regression and TF‑IDF embeddings.
* I started by importing the data and then using the TF‑IDF vectorizer on the train dataset to fit and transform the text into embeddings.
* Next, I brought in the test data, and that’s where I noticed extra columns. They were empty but existed in the CSV because of some additional commas. So, I dropped those columns.
* After cleaning that up, I tried transforming the test data and saw that the label was taken in as a string, which caused an error when I fed it into the Logistic Regression model.
* Finally, I converted the label to integer format, which matched how the model was trained, and that resolved the error.

**def model\_predict (trained\_model, x\_test,y\_test,tf-idf,data=’train’):**

* I wrote a function that takes in arguments like the trained\_model, x\_test, y\_test, tf\_idf, and a parameter data="train".
* The purpose of data="train" is basically to check if we’re dealing with the training data or not. For the training set, we don’t need to do an extra transform since we already used fit\_transform() for it. But for test data, we only do transform() (not fit\_transform()).
* Inside the function, I predict labels, then calculate precision, recall, accuracy, confusion matrix, and F1 score—all the usual suspects for classification metrics.
* The main point in the function is making sure I handle the data transformation correctly based on that data parameter. If it’s train data, I skip the transform part because it’s already been handled during training; otherwise, I just transform the text.

**Part 2: Unsupervised Learning**

**(a): tf-idf embeddings and clustering**

* Since I’d already worked with K‑Means before, it was fairly easy to get started.
* First, I created the TF‑IDF embeddings only from the test data (because we assume no access to the train data for unsupervised clustering).
* After that, I set up the KMeans algorithm with 2 clusters, because we have two potential classes (0 and 1).
* Then, I made a subsample of 50 examples labeled 0 and 50 examples labeled 1 from the test dataset (total 100 rows) and used that to figure out which cluster corresponds to which class label.
* Once I determined the cluster-to-label mapping, I created the predicted labels for the entire test set based on the cluster assignment.
* Finally, I computed the usual metrics: accuracy, precision, recall, and F1 score, by comparing these predicted labels with the true labels.

**(b): Clustering with Sentence Embeddings**

* The process was essentially the same as the TF‑IDF approach, except I used Sentence Transformers for embeddings.
* I loaded the Sentence Transformer model, which was brand new to me. One thins I loved about it is I can use transformer to convert text into more context-aware embeddings.
* After encoding the text using the model, the rest of the steps mirrored the TF‑IDF clustering approach: set KMeans with 2 clusters, create a subsample of labeled data, map clusters to labels, and compute the usual metrics.

So when creating the Sentence Embeddings code, I got the idea of creating a function which will reduce the redundancy in the code. The function which I created is defined below:

**def cluster\_and\_evaluate(test, embedding\_type="tfidf"):**

* I basically followed the same steps for both TF‑IDF and Sentence Transformer embeddings within the function:
* Perform clustering with KMeans set to 2 clusters.
* Create a subsample of 50 examples each for labels 0 and 1.
* Map each cluster to the corresponding label based on the majority in that subsample.
* Compute the accuracy, precision, recall, and F1 scores using the predicted labels.

Where They Differ:

* The main difference was in how the embeddings got created:
  + TF‑IDF Approach: I used the TF‑IDF vectorizer directly on the text data, which transforms each document into a matrix of TF‑IDF features.
  + Sentence Transformer Approach: I loaded the sentence-transformer model, then used it’s encode method on the text to get the embeddings. After that, I just converted those embeddings into a list/array format so I could run KMeans on them.

And in this function when passing the argument if the embedding type was selected as tf-idf then embeddings were created in tf-idf format, and if argument was passed as sentence\_transformer then the embeddings were created from the sentence transformer embeddings.

**Results and Analysis:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Learning Method** | **Embeddings Used** | **Accuracy** | **Precision** | **Recall** | **F1 Score** | **Confusion Matrix** |
| LR Model (Supervised Learning) | TF-IDF (Test Data) | 0.97 | 0.93 | 0.98 | 0.95 | [[6498 275]  [ 77 3416]]  (TN: 6498, FP: 275, FN: 77, TP: 3416) |
| LR Model (Supervised Learning) | TF-IDF (Train Data) | 0.98 | 0.94 | 0.99 | 0.97 | [[58359 1950]  [ 170 31915]]  (TN: 58359, FP: 1950, FN: 170, TP: 31915) |
| K-Means Clustering (Unsupervised Learning) | TF-IDF | 0.66 | 0.50 | 0.61 | 0.55 | [[4608 2165]  [1361 2132]]  (TN: 4608, FP: 2165, FN: 1361, TP: 2132) |
| K-Means Clustering (Unsupervised Learning) | Sentence Transformer | 0.73 | 0.57 | 0.89 | 0.70 | [[4415 2358]  [ 372 3121]]  (TN: 4415, FP: 2358, FN: 372, TP: 3121) |

* Looking at the table, the best performance comes from the LR model (supervised learning) on train data, where it achieves near-perfect accuracy, recall, and F1 score. But comparing performance on train data doesn’t make much sense, so we focus on the test data instead.
* On the test data, the LR model still does an amazing job with an accuracy of 0.97, precision of 0.93, recall of 0.98, and an F1 score of 0.95. Just based on these scores alone, it’s clear that Logistic Regression is handling the task exceptionally well.
* Now, when we contrast this with unsupervised learning using TF‑IDF embeddings, we see a massive drop across all metrics. False Positives (FP) and False Negatives (FN) increase significantly, which means the clustering is struggling to separate misinformation effectively. This makes sense because TF‑IDF embeddings alone, without supervision, don’t capture enough semantic meaning—they just rely on frequency-based vector space clustering, which isn’t ideal for distinguishing misinformation.
* However, when we switch to sentence transformer embeddings instead of TF‑IDF, there’s a noticeable improvement across the board. The biggest boost is in recall, which jumps to 89%, meaning that the model is much better at identifying misinformation compared to TF‑IDF.
* This happens because sentence transformers use a pretrained model with millions of parameters, allowing them to capture semantic similarity between words, unlike TF‑IDF, which just works with raw term frequency. This allows similar words to form better proximity in the vector space, improving clustering performance.
* But even with sentence embeddings, unsupervised learning still falls short of supervised learning. That’s expected supervised learning uses actual class labels, which always gives better scores because the model learns directly from labeled data instead of relying solely on vector space clustering.
* However, the model with the supervised learning with the TF-IDF embeddings will fall short if it sees the set of words/word in which it was not trained on. In such cases, the transformer embeddings model would be the best bet because it being pre-trained, it would’ve seen that word or data and be able to help us predict the result for us.

*Insights based on the Confusion matrix, and how they classify and misclassify misinformation:*

* TF-IDF Clustering: This model over-predicts misinformation, leading to a high number of false positives (2165 FP). This means it incorrectly classifies a lot of non-misinformation as misinformation. Additionally, false negatives (1361 FN) indicate that it also misses a significant number of misinformation cases.
* Sentence Transformer Clustering: This model performs much better in detecting misinformation. The false negative rate drops significantly (FN: 372), meaning it's catching way more actual misinformation cases. However, the false positive rate (FP: 2358) is still high, which means it's still misclassifying some legitimate content as misinformation.
* Supervised Learning (Logistic Regression): Both TF-IDF and sentence transformer clustering methods fall short of the high precision and recall of the LR model. The LR model does the best job in balancing false positives and false negatives, ensuring both high accuracy and reliability.

**Conclusion:**

Overall, Logistic Regression with TF‑IDF embeddings performed the best, achieving high accuracy and recall, but its major limitation is that it would struggle with unseen words since it only learns from the training data. Unsupervised learning with TF‑IDF embeddings performed poorly, failing to separate misinformation effectively. However, using sentence transformer embeddings significantly improved recall, proving that pretrained models can use its vast pre trained data to help us achieve the desired results. If the goal is high accuracy on known data, supervised learning with TF‑IDF is ideal, but for handling new, unseen patterns, transformer embeddings are the better choice.

**Problems faced:**

The only problem I faced was understating why when I was using the test file to test the data for the supervised learning. The issue was, class label in the test data was not integer but string, so I had to convert them to string to get the required result. Now it doesn’t make difference to the naked eye, but the model cannot take that if it was trained on integer data. The problem wasn’t great or anything that hindered my approach but was something that helped me take some analysis.

This problem didn’t trouble me when I was working with the test data for the unsupervised learning, as I had already learnt about the data issue with supervised learning.