

CS M148 Final Project Report: Twitter Financial News Sentiment

Skylesha Marcel, Jacob Kadari, Euleena Trinh, Nathan Truong

Report:

i. The data set your team used

For our project, we used a Twitter Financial News Sentiment dataset from Hugging Face. It consists of tweets and headlines that are financial-related, which were pre-annotated for their sentiment. The label structure is:

- 0 - Bearish
- 1 - Bullish
- 2 - Neutral

For our project, we used the provided splits `sent_train.csv` and `sent_valid.csv`, which had an original size of 9938 rows for the training set and 2486 rows for the testing set.

Once we created the embeddings and class-balanced the dataset, we ended up with 5365 training rows and 2388 testing rows.

ii. The overview of the problem your team addressed

The objective of the project is to predict a sentiment label (Bearish / Bullish / Neutral) given a financial-related tweet or headline. This is a classic multi-class classification problem where the input is unstructured text, and the output is the sentiment classification of it.

iii. The key methodology that worked to address the problem with explanations as to why

The key methods that were used to address the problem began with text preprocessing, an important step to ensure the data was cleaned and normalized to allow our models to train effectively. Additionally, we balanced the classes due to a severe class imbalance to ensure the model was not biased towards the majority class. We chose not to use SMOTE for creating synthetic data for the minority classes, as it did not improve model performance. Next, FinancialBERT was applied to the cleaned data to convert the text into dense numeric vectors. These vectors are high-dimensional numerical representations of the meanings of each sentence by mapping similar meanings together. We chose not to use other embedding models, such as Bag-of-Words, as these methods lost much of the text's meaning, while FinancialBERT captured

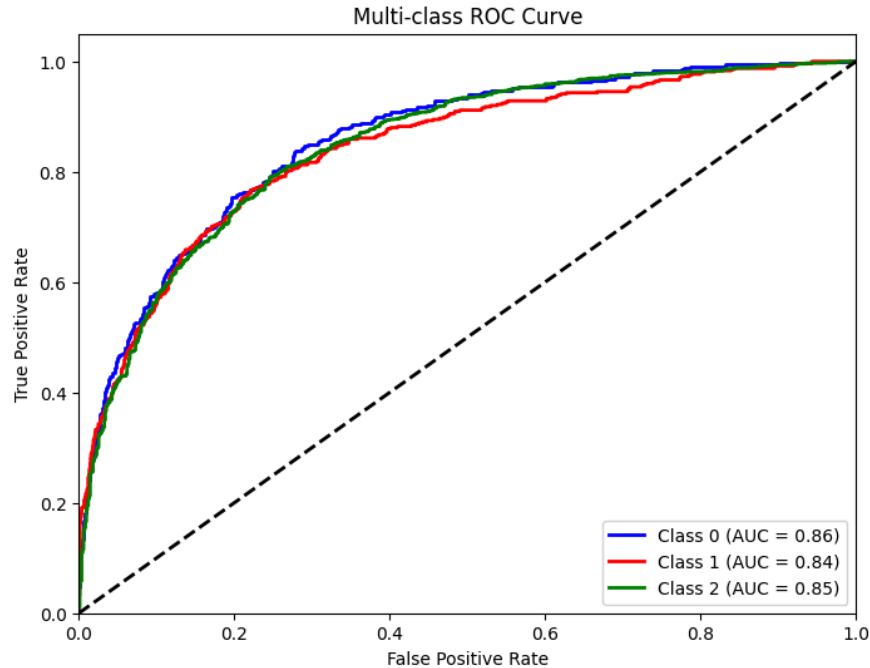
nuances such as sarcasm, context, and domain-specific financial language. Furthermore, FinancialBERT is a specific embedding model that was specifically pre-trained on financial documents and news, thus allowing it to naturally understand the associated meaning with Bearish, Bullish, and Neutral. To reduce the high-dimensionality of the dataset, Principal Component Analysis (PCA) was applied to reduce the noise and unnecessary features. This significantly improved our models' performances as it minimized overfitting to noisy, low-variance features, therefore focusing on the most important patterns. Finally, our main model that performed the best is a random forest. This ensemble method performed the best with its ability to naturally find complex, nonlinear patterns compared to other models evaluated, such as logistic regression. Additionally, random forests can capture higher-order feature interactions within the PCA components, which is significant with the dense embeddings of the text. Furthermore, the random forest required much less hyperparameter tuning than a neural network. Overall, this model performed the best in generalizing to the unseen test data.

iv. Results including cross-validation used and evaluation metrics and conclusions such as why you chose the key method and its limitations

For our random forest model, we used RandomizedSearchCV to run 10-Fold Cross Validation to hyperparameter tune our model. Afterwards, we tested our model and achieved the following results:

Random Forest Evaluation Metrics			
	Precision	Recall	F1 Score
Class 0	0.69	0.74	0.71
Class 1	0.77	0.69	0.73
Class 2	0.75	0.79	0.77

Our random forest model generalized well to unseen data, achieving strong class-wise discrimination performance. We had a prediction accuracy of 0.7387 and an overall F1 score of 0.7385. Specifically, the model attained an AUC of 0.86 for class 0, 0.84 for class 1, and 0.85 for class 2. The consistently high AUC values across all classes suggest that the model learned meaningful and robust patterns in the data rather than overfitting to the training set. Also, this indicates that the model is able to reliably distinguish each class from the others across a wide range of decision thresholds, demonstrating balanced and consistent predictive performance rather than favoring a single class.



We selected the random forest as our primary model because it consistently achieved the strongest performance across all evaluation metrics when compared to logistic regression and neural networks. With its highest macro F1 score, accuracy, and AUC values, the random forest is best distinguished amongst the three sentiment classes.

Unlike logistic regression and neural networks, which assume more defined decision boundaries, random forest was able to capture the complex, non-linear relationships and feature interactions that appeared in the high-dimensional embeddings. This flexibility allowed it to better separate overlapping sentiment classes that the other models failed to differentiate. Furthermore, neural networks require extensive hyperparameter tuning and are prone to overfitting in a smaller data set. Overall, the random forest model generalized well to unseen data and provided the most robust and interpretable performance amongst all models considered.

Restrictions of our modeling:

- Imbalanced Class Distribution: The dataset is highly imbalanced, which can still affect minority-class performances even when class balancing is applied to the dataset.
- Sensitivity to High-Dimensional Embeddings: Although PCA reduced dimensionality, embedding-based features can still present noise and may remove some fine-grained semantic information.
- Computational Cost: Generating and processing FinancialBERT embeddings is computationally expensive and sensitive to preprocessing choices.
- Generalization Risk: While PCA improved generalization, tree-based models can still overfit if dimensionality reduction or regularization is insufficient.

v. How to use the code for your project on the data set

Step 1: Install required packages

- Install the following Python packages prior to running the project: Pandas, NumPy, scikit-learn, PyTorch, Transformers, NLTK, Emoji, and other necessary packages listed in the Notebooks.

Step 2: Load the data

- Load the provided sent_train.csv and sent_valid.csv files, each containing the text and corresponding sentiment label columns.

Step 3: Text preprocessing

- Preprocess the raw text by removing URLs, emojis, and punctuation, normalizing whitespace, converting text to lowercase, removing stopwords, and applying custom financial-domain token removals. This preprocessing step ensures that the text is cleaned and standardized prior to embedding generation.

Step 4:

- Embeddings can be obtained in one of two ways:
 - Using our Preprocessing Notebook, run FinancialBERT on the preprocessed text to generate embeddings and save them as training_embed.csv and valid_embed.csv.
 - Load previously generated embedding files (training_embed.csv and valid_embed.csv). This was the approach used during the project check-ins.
- Finally, we then class balance the data, generating balanced_training_embed.csv

Step 5: Train the model

- Using the generated balanced_training_embed.csv, train the Random Forest model with the hyperparameter optimization search. This will automatically run a 10-Fold Cross Validation to tune the hyperparameters

Step 6: Evaluate:

- Evaluate model performance by computing accuracy, weighted precision, weighted recall, weighted F1 score, and the confusion matrix on the testing dataset.

Appendix:

- Explain the exploratory data analysis that you conducted. What was done to visualize your data and split your data for training and testing?**

We examined the structure of the dataset to verify that it consisted of two primary fields: the financial text (tweet or headline) and the corresponding sentiment label. We confirmed that the provided training and testing splits aligned with expectations and were consistent across

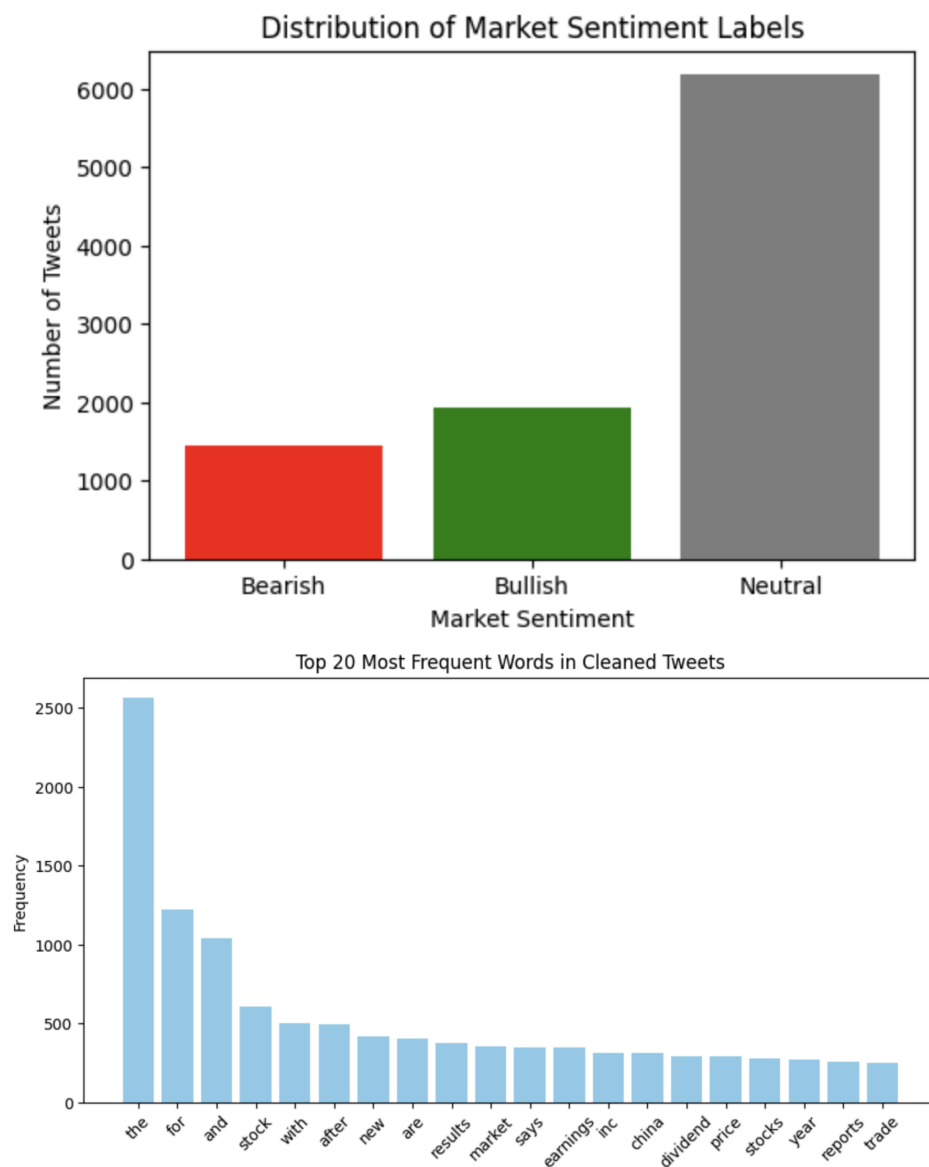
check-ins. As part of the exploratory data analysis, we evaluated the distribution of sentiment labels in the training set and observed a clear class imbalance, with the following counts:

label 0 (Bearish) = 1442

label 1 (Bullish) = 1923

label 2 (Neutral) = 6178

To further confirm this imbalance, we computed label proportions, which showed that the Neutral class comprised a majority of the dataset. These exploratory steps informed later modeling decisions, particularly the need to address class imbalance in classification tasks.



ii. What data pre-processing and feature engineering (or data augmentation) did you complete on your project?

We implemented a standardized text-cleaning pipeline tailored for financial tweets and headlines. This preprocessing included removing all URLs, emojis, and non-text noise, converting all text to lowercase, normalizing whitespace by replacing multiple spaces with a single space, removing stopwords, and filtering out finance-related filler tokens using a custom removal list. These steps reduced noise while preserving semantically meaningful financial language prior to model input.

We experimented with multiple feature representations throughout the project. These included simple engineered “signal” features such as the presence of sentiment-related financial keywords (for example: buy, sell, gain, loss, upgrade, downgrade) for logistic regression. We also extracted part-of-speech (POS) count features, including the number of nouns, adjectives, and verbs, which were used in the regression check-in. Finally, for classification tasks, we used dense sentence-level embeddings generated from BERT-based models, specifically FinancialBERT, to capture semantic and contextual information in the text.

At a high level, BERT-related models take our raw, clean text and produce a single 768-dimensional vector used to represent the semantic context for our data. The first step involved tokenizing the words, which takes text and splits it into tokens that map to numerical values. After tokenization, PyTorch was then required as BERT utilizes a deep neural network in order to generate the embeddings through a forward pass in a single aggregate sentence.

Lastly, we fixed the issue of class imbalance by randomly sampling 2000 observations from all the observations that were labeled Neutral. With the newly balanced training dataset, we have 2000 for Neutral, 1923 for Bullish, and 1442 for Bearish.

iii. How was regression analysis applied in your project? What did you learn about your data set from this analysis and were you able to use this analysis for feature importance? Was regularization needed?

Regression analysis was applied as an exploratory modeling exercise rather than as a solution to the final sentiment classification task. Specifically, we performed linear regression using part-of-speech (POS)–based predictors (the number of nouns, adjectives, and verbs) to predict tweet word count as a continuous response variable. This allowed us to examine how basic linguistic structure relates to text length in financial tweets.

The linear regression model achieved strong and consistent performance, with a training R^2 of 0.869 and a testing R^2 of 0.872, along with similar MAE and RMSE values across both splits. These results indicate good generalization and no evidence of overfitting. We additionally

applied Ridge regression with cross-validated regularization and tested a range of α values, selecting $\alpha = 0.1$. The Ridge model produced nearly identical performance and coefficients to the standard linear regression model, indicating that regularization was not necessary due to the small number of predictors and the stability of the model.

From this analysis, we learned that POS counts provide a strong and interpretable representation of tweet length, and the regression coefficients offered a limited form of feature importance by quantifying the contribution of nouns, adjectives, and verbs to overall word count. However, this regression approach was not well aligned with the main project goal. Sentiment classification is a categorical task, and POS-based features capture grammatical structure rather than semantic polarity. As a result, we treated this regression analysis as a modeling exercise to better understand the dataset rather than a method used in the final sentiment classification pipeline.

iv. How was logistic regression analysis applied in your project? What did you learn about your data set from this analysis and were you able to use this analysis for feature importance? Was regularization needed?

Logistic regression was applied as a binary classification task to explore whether simple engineered features could separate neutral sentiment from non-neutral sentiment. We reformulated the original three-class sentiment labels into a binary outcome, where “Neutral” was treated as one class and “Not Neutral” (Bearish or Bullish) as the other. The model used a single engineered predictor, `has_keyword`, which indicates the presence of sentiment-related financial keywords in the text.

The fitted logistic regression model produced an intercept of 0.7549 and a coefficient of -1.0194 for the `has_keyword` feature. This coefficient served as a direct and interpretable measure of feature importance, indicating that the presence of these keywords was associated with a lower probability of the Neutral class. For evaluating the model, we used 5-Fold Cross Validation on the training set. The fold AUC values ranged from 0.550 to 0.570, and fold accuracies were 0.627. These results indicate that while the model was able to capture some signal, its discriminative ability was limited. The high recall and low specificity suggest that the classifier tended to predict the majority Neutral class, reflecting the underlying class imbalance in the dataset.

To finally test the model on unseen data, we used the testing set, the model achieved an accuracy of 0.6750, a recall (true positive rate) of 0.9183, a specificity (true negative rate) of 0.2117, and an area under the ROC curve (AUC) of 0.562.

For regularization, we used a standard logistic regression configuration with the liblinear solver and did not perform an extended regularization sweep. This choice was intentional, as the model

contained only a single engineered feature, and there was no evidence of overfitting or instability that would require additional regularization. Overall, this analysis demonstrated the limitations of simple, surface-level features for sentiment classification and motivated the use of embedding-based representations in later stages of the project.

v. How were KNN, decision trees, or random forest used for classification on your data? What method worked best for your data and why was it good for the problem you were addressing?

We explored several classification approaches such as KNN, decision trees, and random forests, to model sentiment classification on our dataset. KNN was limited by the high dimensionality of the embeddings and overlapping nature of sentiment analysis, reducing its effectiveness to differentiate between the classes. Also, single decision trees are able to model highly nonlinear relationships, but are not as robust compared to random forests. Thus, we primarily chose to use random forest models trained on FinancialBERT embedding features.

Our baseline Random Forest was trained directly on the raw 768-dimensional embeddings using:

- `n_estimators = 250`
- `max_depth = 16`
- `class_weight = balanced`

On the training set (multi-class), this baseline model performed extremely well (Accuracy = 0.87, Weighted F1 = 0.88), but it did not generalize to unseen data, indicating overfitting. Using 5-Fold Cross Validation, we observed:

- Mean AUC = 0.541
- Mean Accuracy = 0.571

To improve our model further, we used `RandomizedSearchCV` with a 10-Fold Cross Validation to test many different hyperparameters configurations such as max depth, minimum samples per leaf, max features, etc. Furthermore, to address the overfitting from using high-dimensional embeddings, we included PCA as a hyperparameter to tune within the pipeline for `RandomizedSearchCV`. The best hyperparameters found were:

- `n_estimators: 400`
- `min_samples_split: 300`
- `min_samples_leaf: 100`
- `max_features: log2`
- `max_depth: 10`
- `class_weight: balanced`
- `ccp_alpha: 0.0001`

- `pca__n_components`: 150

Ultimately, by using PCA for dimensionality reduction and performing extensive hyperparameter tuning, the random forest model worked best for our complex, nonlinear data.

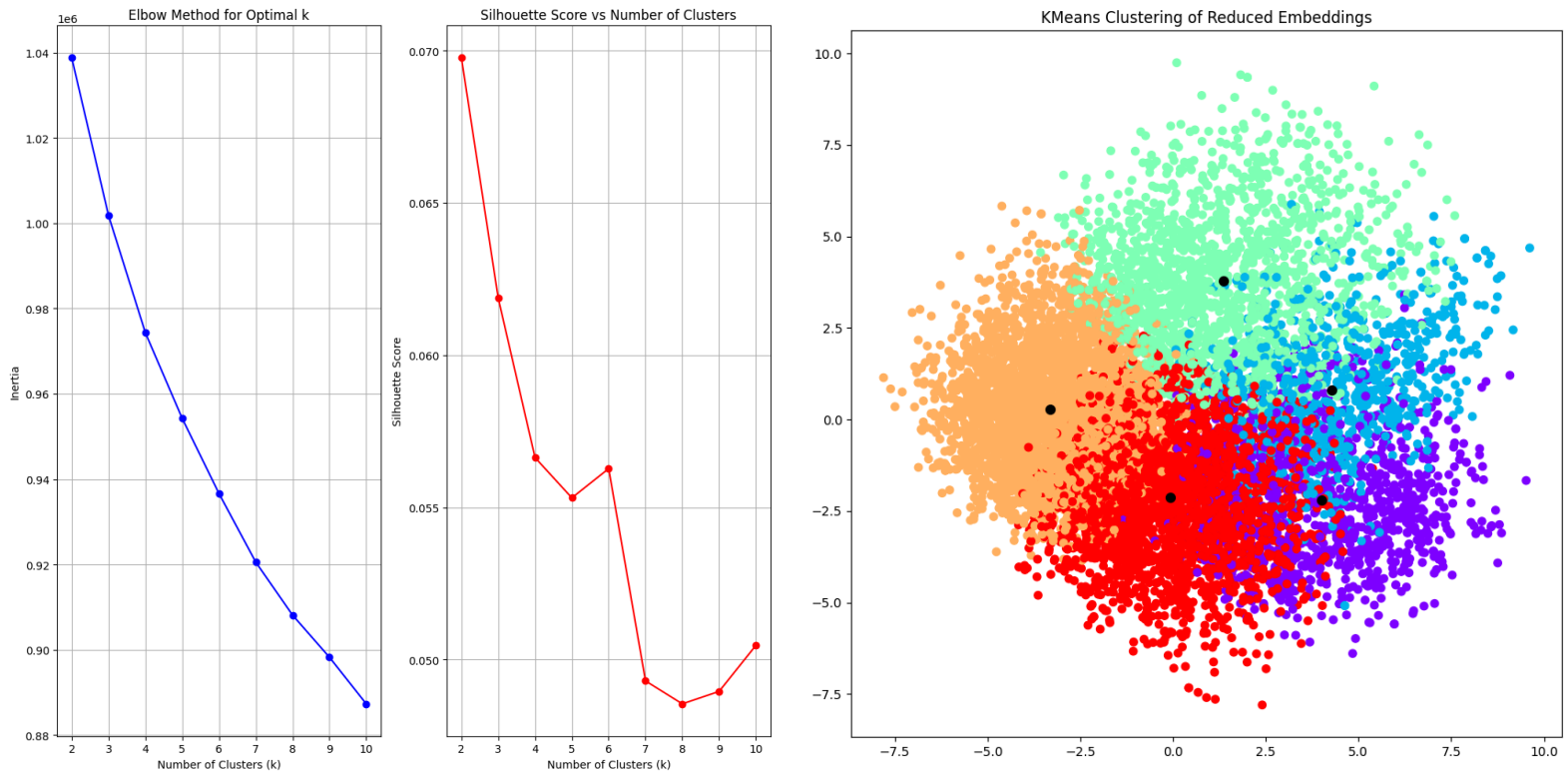
vi. How were PCA and clustering applied on your data? What method worked best for your data and why was it good for the problem you were addressing?

We applied PCA and clustering as unsupervised learning techniques to explore the underlying structure in the FinancialBERT embedding representations of tweets and headlines. Because each text sample is represented by a high-dimensional (700+ dimensional) embedding vector, directly clustering in the original space would lead to unreliable distance calculations due to noise and redundancy across dimensions.

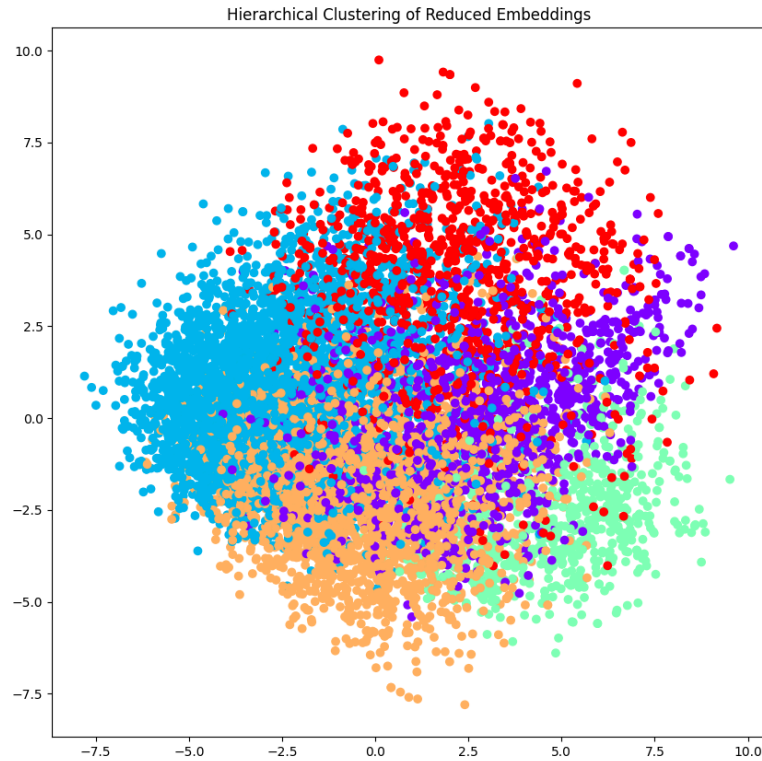
To address this, we first applied PCA to reduce dimensionality while preserving most of the meaningful variance in the data. We used `n_components = 100`, reducing the embedding matrix from a high-dimensional space to a shape of (9543, 100). These 100 components explained approximately 84.5% of the total variance. The scree plot showed a sharp decline in variance across the first several components, followed by a flattening curve, indicating diminishing returns beyond 100 components. This confirmed that PCA effectively compressed the embeddings while retaining their core structure and improving the reliability of distance-based methods.

After PCA, we applied three clustering techniques: K-Means, hierarchical agglomerative clustering (Ward linkage), and HDBSCAN.

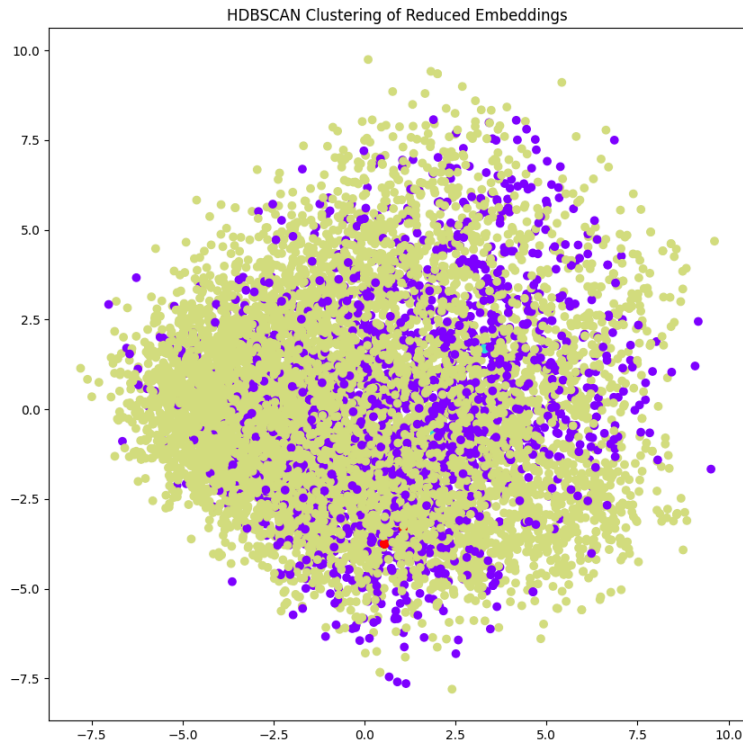
For K-Means clustering, we evaluated the number of clusters using both the elbow method (based on inertia) and silhouette scores. The elbow method suggested $k = 5$, while the highest silhouette score occurred at $k = 2$, which corresponds to a coarse split between neutral-like and non-neutral-like behavior. We selected $k = 5$ to allow for a more granular structure. With this choice, K-Means produced an inertia of 889,864.3 and a silhouette score of 0.0579, indicating substantial overlap between clusters and weak separation. However, looking at the visualization of the KMeans clustering, there seems to be defined clusters.



We next applied hierarchical agglomerative clustering using Euclidean Distance and Ward linkage. This method yielded a silhouette score of 0.0368, which further supported the conclusion that strong cluster separation was not present at this level of granularity.



Finally, we used HDBSCAN, a density-based clustering method that can identify clusters of varying density and label ambiguous points as noise. HDBSCAN produced 3 clusters (excluding noise) and labeled 2470 out of 9543 tweets (25.88%) as noise. The non-noise cluster sizes were 17, 7029, and 27. When noise points were excluded, HDBSCAN achieved a higher silhouette score of 0.1812, indicating that a small subset of tweets formed denser, more coherent clusters, while a large portion of the data remained ambiguous.



To compare clustering consistency, we computed the Adjusted Rand Index (ARI) between methods. The ARI values were 0.028 for K-Means vs. HDBSCAN, 0.326 for K-Means vs. hierarchical clustering, and 0.012 for HDBSCAN vs. hierarchical clustering. These results show moderate agreement between K-Means and hierarchical clustering, while HDBSCAN produced substantially different assignments due to its density- and noise-based structure.

HDBSCAN was the best method being the most informative clustering approach because it explicitly identified ambiguous or noisy tweets rather than forcing all points into clusters, which better reflects the continuous and overlapping nature of sentiment in financial text embeddings.

vii. Explain how your project attempted to use a neural network on the data and the results of that attempt.

We attempted to use a feedforward neural network to perform three-class sentiment classification (Bearish, Bullish, Neutral) using FinancialBERT embeddings. Because the original embeddings are high-dimensional, we first applied Principal Component Analysis (PCA) to reduce the feature space to 100 components, which retained approximately 84.5% of the total variance and provided a more compact and manageable input for the neural network.

The pipeline followed these steps: cleaned text was converted into FinancialBERT embeddings, PCA was applied to extract the top 100 components, and the reduced feature vectors were then used to train the neural network.

The neural network architecture consisted of an input layer with 100 PCA features, followed by two hidden layers with 128 and 64 neurons using ReLU activation functions, and an output layer with 3 logits, one for each sentiment class. The model was trained using cross-entropy loss and optimized with stochastic gradient descent (SGD) over 100 epochs, with the learning rate selected through hyperparameter tuning.

On the testing dataset, the neural network achieved an accuracy of 0.6558, weighted precision of 0.4300, weighted recall of 0.6558, and weighted F1 score of 0.5194. While the overall accuracy appeared moderate, the confusion matrix revealed that the model heavily favored predicting the majority class. This indicates that the neural network struggled to meaningfully separate the sentiment classes and instead learned to optimize accuracy by predicting the most frequent label, a common failure mode when training neural networks on imbalanced datasets.

We learned that this experiment showed that although neural networks can work well with embedding-based features, class imbalance combined with relatively simple training constraints can cause the model to collapse toward the majority class. A stronger version of this approach would require techniques such as class-weighted or focal loss functions, balanced sampling strategies, stronger optimizers, mini-batch training, and additional regularization to improve minority-class performance and overall generalization.

We also hypothesize that the neural network underperformed in comparison to the random forest, primarily due to substantially less hyperparameter tuning. The random forest benefited from our extensive hyperparameter tuning when adjusting for depth, size, and PCA variability. When running our neural network, however, it was trained with more limited configurations. With comparable hyperparameter tuning, it is plausible that the neural network may perform similarly, if not better, than the random forest, given its ability to model nonlinearity.

viii. Give examples of hyperparameter tuning that you applied in preparing your project and how you chose the best parameters for models.

Hyperparameter tuning was performed throughout the project with the primary goal of improving generalization on the unseen data rather than simply optimizing performance on the training set. Below are the main examples of hyperparameter tuning and how the final parameters were selected.

Dimensionality of PCA: We selected 100 principal components for PCA, resulting in a reduced feature matrix of shape (9543, 100) while retaining approximately 84.5% of the total variance. This choice was guided by the scree plot, which showed that the first 100 components captured most of the meaningful structure in the data, with diminishing returns beyond that point. This allowed for significant dimensionality reduction without sacrificing important information.

Random Forest Hyperparameters and PCA Integration: For the Random Forest classifier, we initially used `n_estimators = 250`, `max_depth = 16`, and `class_weight = 'balanced'`. Although this configuration produced strong training performance, it showed weak generalization on the validation set, indicating overfitting. To address this, we introduced `RandomizedSearchCV` that optimized the number of PC components used from PCA and other hyperparameters. This change substantially improved validation performance, including accuracy, F1 score, and per-class AUC values.

Clustering Parameter Selection: Although clustering is unsupervised, we used quantitative metrics to guide parameter selection. For K-Means, we used the elbow method and silhouette scores to evaluate different values of `k`. The elbow method suggested `k = 5`, while silhouette scores peaked at `k = 2`. We selected the final value of `k` based on whether a coarse or finer-grained clustering structure was desired. For HDBSCAN, we set `min_cluster_size = 15` to prevent the formation of very small, unstable clusters and to allow the model to identify noise points effectively.

For neural networks:

- **Learning Rate:** The most important hyperparameter tuned for the neural network was the learning rate. We evaluated three learning rates: 0.001, 0.01, and 0.1. A learning rate of 0.001 resulted in very slow learning and underfitting, with minimal improvement over training. A learning rate of 0.1 led to unstable training behavior, including overshooting and a lack of convergence. The learning rate of 0.01 provided the best balance between training stability and validation performance and was therefore selected as the final value.
- **Number of Training Epochs:** The neural network was trained for 100 epochs. This value was chosen by monitoring training and validation performance and continuing training until validation metrics stopped improving. Training beyond this point did not lead to additional gains, so 100 epochs was selected as the final setting.

Links To All Google Colab Notebooks:

 [CS M148 Project Preprocessing](#)

 [CS M148 Project Exploratory Analysis](#)

- CS M148 Linear Regression
- CS M148 Project Logistic Regression
- CS M148 Project PCA and Clustering
- CS M148 Random Forest
- CS M148 Project Neural Network