CIS 4190/5190: Applied Machine Learning Spring 2025

Milestone 2

Team Members: Ashay Katre, Vinod Ghanchi, Shashank Kambhatla *Project: News Source Classifier*

1 Summary

This project addresses the task of binary news source classification using only article headlines. Given the known political and editorial differences between outlets, particularly Fox News and NBC, our goal was to design machine learning models that could identify subtle linguistic cues to accurately predict the source of a headline.

Our dataset consisted of approximately 5,000 headlines, curated by scraping from the provided list of URLs and augmented with additional samples to ensure class balance and coverage diversity. We performed a thorough preprocessing pipeline that included lowercasing, removal of standard and custom stopwords, and headline cleaning. The custom stopword list was crafted to eliminate branding-related or layout-specific terms (e.g., "fox", "nbc", "breaking") that might lead to label leakage.

We experimented with several classical models — including Logistic Regression, Support Vector Machines, Multinomial Naive Bayes, Random Forests, and XGBoost — and paired these with multiple feature extraction strategies such as TF-IDF, CountVectorizer, and HashingVectorizer. We also conducted a semantic embedding experiment using Sentence-BERT (MiniLM), which, despite being more advanced, underperformed relative to simpler TF-IDF approaches, likely due to the short length and limited context in headlines.

Each pipeline was systematically optimized using GridSearchCV with cross-validation, and evaluated using a consistent framework with metrics including accuracy, F1-score, and ROC AUC. Our best-performing model — Support Vector Machine with an RBF kernel based on TF-IDF bigrams and — achieved approximately 80% accuracy, outperforming all baselines. Exploratory experiments around n-gram patterns, vectorizer types, and semantic embeddings further informed and validated our design choices.

Ultimately, we found that classical models with rich lexical features like bigram TF-IDF remain highly effective in short-text classification tasks like this, often outperforming deeper semantic models on practical metrics.

2 Core Components

2.1 Data Collection and Cleaning

Procedure and Protocol for Data Collection

The provided CSV contained 3,805 article URLs from FoxNews and NBC. To improve generalizability and enhance class balance, we expanded the dataset through manual web scraping using requests and BeautifulSoup, collecting an additional 700 NBC and 500 FoxNews headlines. Each URL was parsed to extract the headline, using outlet-specific HTML structures. For example, FoxNews headlines were often located within an <h1> tag with class headline speakable, while NBC's layout varied more and required multiple fallback strategies (e.g., meta tags, alt text, or specific <h1> tags). To ensure robustness, we also verified HTTP status codes, removed duplicate headlines, and filtered out pages where scraping failed. Our final raw dataset consisted of 5,200 headlines - evenly balanced between the two outlets.

Curation and Cleaning

The scraped headlines were then cleaned using a structured NLP pipeline:

- Lowercasing: All text was converted to lowercase for normalization

- **Stopword Strategy**

    1. **Default Stopword Removal** - We removed standard English stopwords ("the", "is", "and", etc), that add syntactic structure but carry little semantic value, using nltk.corpus.stopwords.

    2. **Custom Stopword List** - We faced an issue during our project where some words in our headlines were non-informative but strongly correlated with outlet branding, layout cues, or visual presentation and not actual content. We built a custom stopword list by manually adding words that are frequent but carry no discriminative power, often used by both outlets in headings or layout tags. These included direct mentions of the outlet (leak label info i.e. 'fox' or 'nbc') and formatting or callout words not related to content (i.e. "news", "update", "report", etc). We saved this curated stopword list into a custom_stopwords.txt file to ensure reusability - list was applied consistently across all vectorizers and modeling pipelines.

- **Character Sanitization** HTML tags, escape characters, and extra spaces were removed. Stemming/lemmatization was also applied in certain cases to compare if it improved any performance

  The dataset was stored with the title, outlet (Fox or NBC), label (Fox or NBC), and url columns.

2.2 Model Design

2.2.1 Initial Design Considerations

We began with a **simple baseline model**: Logistic Regression using TF-IDF vectorized unigrams. This was selected for its interpretability, low complexity, and established performance in short-text classification. The baseline helped us establish early expectations, producing an initial test accuracy of approximately 66%, confirming that even a linear model could pick up on surface-level linguistic patterns distinguishing the two outlets. However, we quickly identified areas for potential improvement: the representation (unigrams) might be too shallow, and the model capacity too limited to capture framing or editorial nuance.

However, we quickly identified areas for potential improvement: the representation (unigrams) might be too shallow, and the model capacity too limited to capture framing or editorial nuance.

2.2.2 Feature Representation Experiments

We systematically explored various **text vectorization strategies** to enhance how headlines were encoded:

- **TF-IDF Vectorizer:** Our primary representation. Tuned with different `max_features` (1000 to 5000) and `ngram_range` values. Adding bigrams slightly improved performance by capturing editorial phrasings.

- **CountVectorizer:** Used primarily with Naive Bayes & SVM, where raw term frequency often performs as good as TF-IDF

- **HashingVectorizer:** Explored for its efficiency and fixed memory footprint. Lacks interpretability - so didn't use in our final models.

- **Sentence Embeddings (MiniLM-BERT):** We explored `all-MiniLM-L6-v2` embeddings using the `sentence-transformers` library to encode headlines as dense, semantic vectors. These were evaluated using Logistic Regression and Random Forest. However, results consistently lagged behind TF-IDF-based models, likely due to the short, sparse nature of headlines offering limited context for deep embeddings to leverage.

2.2.3 Modeling and Optimization Process

Each model was paired with its respective feature representation inside a `Pipeline`, and tuned using `GridSearchCV` with k-fold cross-validation. Key models explored:

- **Logistic Regression:** Our baseline and one of the most consistent performers. Tuned via `C`, `solver`, and `max_iter`. Bigrams and increased feature size led to substantial performance gains.
- **Multinomial Naive Bayes:** Paired with CountVectorizer and TF-IDF. Simple, fast, and interpretable—especially effective when class distributions were well balanced. Also used to extract the most predictive tokens per class via `feature_log_prob_`.
- **Support Vector Machines (SVM):** Explored both linear and RBF kernels. While RBF yielded better ROC-AUC scores, it came with longer training times. Bigram TF-IDF with RBF kernel offered near-best performance overall.
- **Random Forest & XGBoost:** Tree-based models were highly effective when combined with TF-IDF. We tuned parameters like `max_depth`, `n_estimators`, and `learning_rate` (for XGBoost),and while these models performed reasonably well (accuracy around 76%), they did not surpass the performance of SVM. Due to their longer training time and less consistent generalization, we did not select them as our final model.

2.3 Evaluation Protocol and Results

To assess model quality and guide model selection, we employed a rigorous evaluation strategy based on multiple performance metrics and controlled cross-validation.

Train-Test Split: We used an 80-20 stratified split of our dataset to ensure both training and test sets maintained class balance between FoxNews and NBC headlines. All model training and tuning were performed exclusively on the training set to avoid data leakage.

Cross-Validation: For all pipelines, we used k-fold cross-validation inside `GridSearchCV` to tune hyperparameters, ensuring our models are robust to overfitting; optimized using average across multiple folds.

Metrics: We evaluated each model using **Accuracy** (to capture overall correctness), **F1-score** (balance precision and recall, especially when performance across classes varied) and **ROC AUC** (to measure model's ability of separating classes across thresholds)

Model Selection: After running multiple combinations of vectorizers and classifiers, we selected our final model based on a combination of test accuracy, F1-score, and ROC AUC. Notably:

- **Logistic Regression with TF-IDF unigrams** served as our baseline, achieving 66% accuracy.
- Adding `ngram_range = (1,2)` lifted performance to 77% accuracy with improved F1-scores.
- **SVM with TF-IDF bigrams** reached up to 78–79% accuracy, but was slower to train.
- **Random Forest and XGBoost with TF-IDF bigrams** also got a very good performance of approximately 76% test accuracy and ROC AUC nearing 0.85.

All evaluations were conducted on the held-out test set to simulate real-world performance. We also used confusion matrices and ROC curves to visualize model behavior and investigate common misclassification patterns. These insights informed final model selection and guided our exploratory investigations.

On the instructor-provided 20-sample mini test set, our models achieved 70% accuracy with Random Forest, 85% with Naive Bayes, and 90% with SVM (all using TF-IDF features). Based on this, we selected SVM as our final model for full test inference.

Table 1: Performance Comparison Across Models and Embedding Techniques

Model	Embedding	Key Parameters	Acc	F1	ROC AUC
Logistic Regression	TF-IDF (Unigram) -Baseline	max_features=100, C=1.0	0.66	0.67	0.74
	TF-IDF (Unigram + Bigram)	max_features=5000, C=1.0	0.77	0.78	0.86
	Sentence-BERT (MiniLM)	384-d vector, solver: saga	0.734	0.74	0.81
Naive Bayes	TF-IDF (Unigram + Bigram)	max_features=5000, alpha=1.0	0.774	0.77	0.854
	CountVectorizer(Unigram + Bigram)	max_features=5000, alpha=1.0	0.775	0.77	0.86
	Hashing Vectorizer	max_features=5000, alpha=1.0	0.736	0.77	0.81
SVM	TF-IDF (Unigram + Bigram)	C=10, kernel=rbf, gamma=scale	0.78	0.77	0.863
	CountVectorizer (Unigram + Bigram)	C=10, kernel=rbf, gamma=scale	0.786	0.79	0.864
	HashingVectorizer	C=10, kernel=rbf, gamma=scale	0.76	0.76	0.84
Random Forest	TF-IDF	Default Params	0.76	0.76	0.842
	TF-IDF (Unigram + Bigram)	n_estimators=200, max_depth=50, min_samples_split=5, max_features = log2	0.76	0.75	0.843
	BERT + GridSearch	n_estimators=250, max_depth=40, max_features = sqrt	0.66	0.66	0.72
XGBoost	TF-IDF (Unigram)	max_features=4000, max_depth=5, lr=0.1	0.71	0.72	0.80
	CountVectorizer(Bigram)	max_features=3000, n_estimators=200, max_depth=5 , lr=0.1	0.732	0.73	0.81
	HashingVectorizer	n_estimators=200, max_depth=5 , lr=0.1	0.71	0.72	0.879

3 Exploratory Questions

3.1 Which vectorizer performs best: TF-IDF, CountVectorizer, HashingVectorizer, or Sentence-BERT?

Question and Motivation: We aimed to evaluate which feature extraction technique best represents short headlines for source classification. Specifically, we compared `CountVectorizer`, `TF-IDF`, `HashingVectorizer`, and semantic embeddings driven s`Sentence-BERT (MiniLM)` to determine their effectiveness across multiple classifiers.

Prior Work / Expectations: From course material and research, we expected TF-IDF or Sentence-BERT to outperform CountVectorizer due to their ability to capture either term rarity or semantic meaning. However, some literature suggests that for short texts, simpler frequency-based encodings like CountVectorizer may be sufficient or even superior due to limited context windows.

Methods: We trained models (Logistic Regression, Naive Bayes, SVM, and Random Forest) on different

vector representations of the same dataset. We tuned each model using GridSearchCV, and evaluated on held-out test sets using Accuracy and ROC AUC.

Results and Updated Beliefs: Contrary to our expectations, TF-IDF consistently outperformed CountVectorizer across all models, especially when bigrams were used. Sentence-BERT underperformed TF-IDF, likely due to the extremely short length of headlines limiting its ability to model contextual semantics. HashingVectorizer worked acceptably in tree-based models but lacked interpretability.

Limitations: We did not explore Word2Vec due to limited dataset size and headline length. Also, Sentence-BERT embeddings were not fine-tuned, which may explain their weaker performance. Additional experiments with contextual finetuning or longer text inputs may shift these results.

3.2 Do bigrams significantly improve classification performance?

Question and Motivation: We hypothesized that editorial framing differences between outlets would appear in multi-word phrases (e.g., "biden admin", "school shooting"). Hence, we explored whether including bigrams would improve performance over unigrams alone.

Prior Work / Expectations: N-gram models are widely used in traditional NLP, and it's known that bigrams capture more semantic content. Prior results in short-text classification support the idea that bigrams offer strong returns on performance in sparse contexts.

Methods: While we isolated unigrams vs bigrams in controlled experiments only for Logistic Regression, we incorporated both `ngram_range=(1,1)` and `(1,2)` into our GridSearchCV pipelines across all models (Logistic Regression, SVM, Naive Bayes, and XGBoost). This allowed the models to select between unigrams and bigrams based on cross-validation performance.

Results and Updated Beliefs: In nearly all cases, the best-performing configurations used `ngram_range=(1,2)`, suggesting that bigrams provided useful signal across models. For example, our Logistic Regression model improved from a baseline of 66% (with default unigrams) to 77% with tuned parameters that included bigrams. SVM and XGBoost models also selected bigram-based TF-IDF as part of their optimal configuration, leading to improvements of 2–4% in both Accuracy and ROC AUC.

Limitations: Bigrams increase feature dimensionality and sparsity, which may hurt performance if the dataset were smaller or more imbalanced. Did not explore trigrams due to runtime concerns, though they may offer marginal gains.

4 Team Contributions

- **Ashay Katre:** Led SVM, Logistic Regression, and Naive Bayes Classifier modeling. Designed exploratory experiments, feature comparison framework, and managed report writing and refinement.
- **Vinod Ghanchi:** Handled data augmentation by Developing web scraping pipeline, and creating scustom stopwords. Focused on Random Forest modeling and hyperparameter tuning. Assisted with preparing visualizations.
- **Shashank Kambhatla:** Worked on XGBoost Classifier, and contributed to EDA and exploratory question evaluations, as well as preparation of slides.

5 Conclusion

TF-IDF with bigrams consistently outperformed other embeddings (including Sentence-BERT), proving that classic lexical features remain highly effective for short headline classification. Support Vector Machines (SVM) with RBF kernel achieved the best overall performance with 80% accuracy and ROC of 0.86. Feature design and preprocessing (e.g., custom stopwords) had as much impact as model complexity, reinforcing the importance of data-centric approaches in applied ML tasks.

Appendix

A. Baseline - Simple Logistic Regression

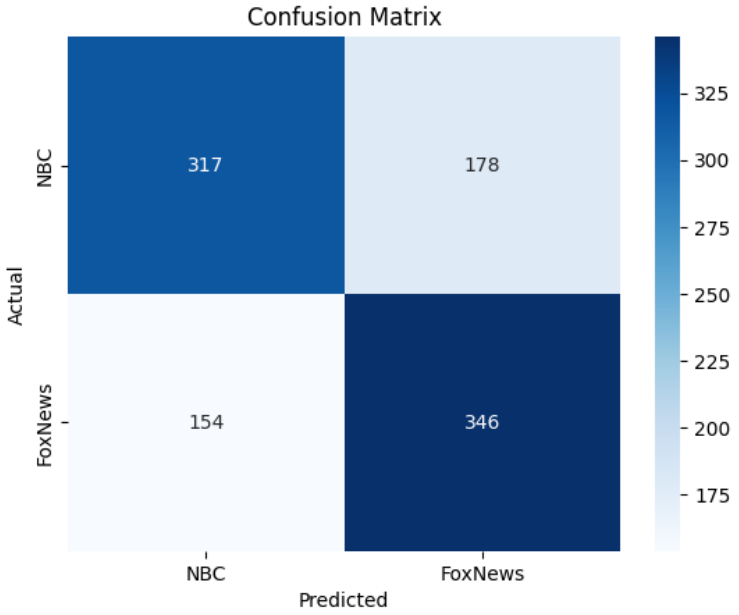


Figure 1: Baseline Logistic Regression Model Confusion Matrix

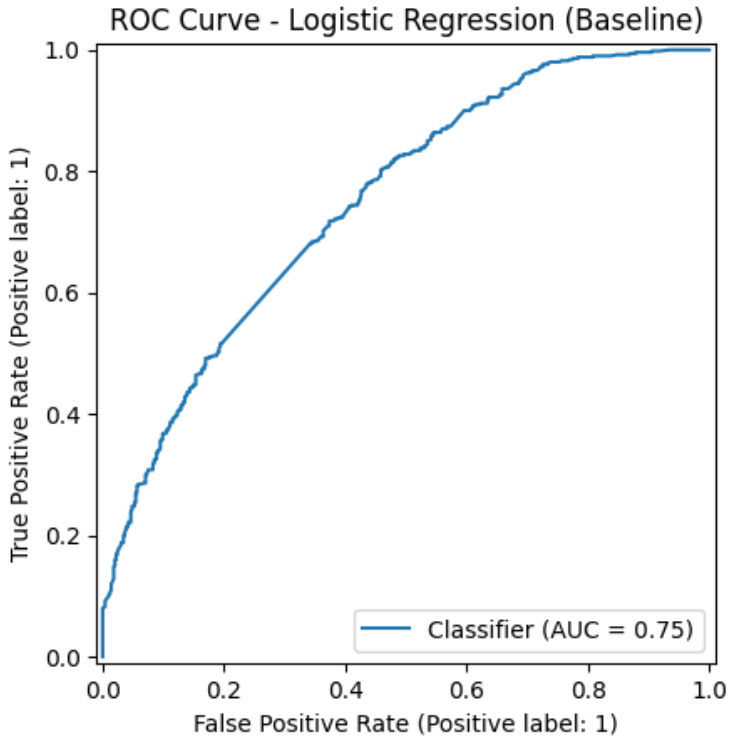
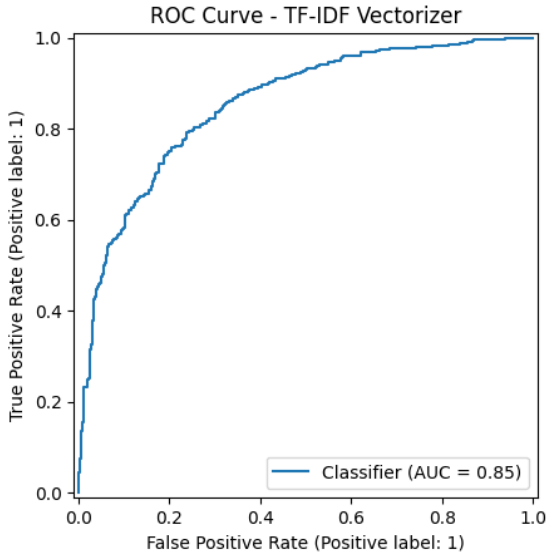
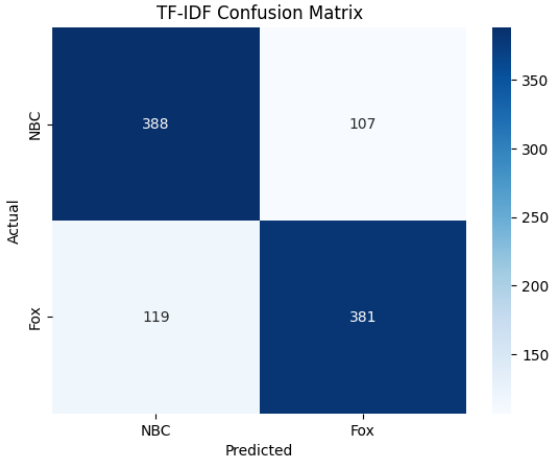


Figure 2: Baseline Logistic - ROC-AUC Curve

B. Naive Bayes

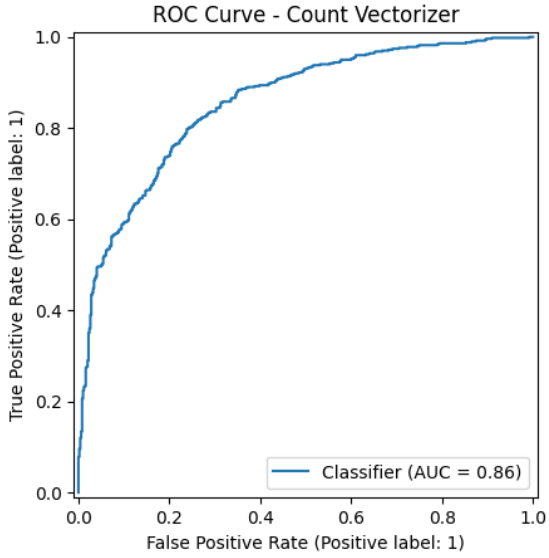


(a) ROC-AUC curve

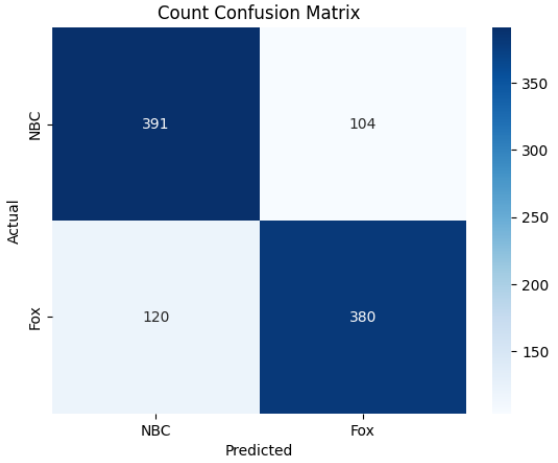


(b) Confusion Matrix

Figure 3: Naive Bayes with TF-IDF Vectorizer: ROC and Confusion Matrix

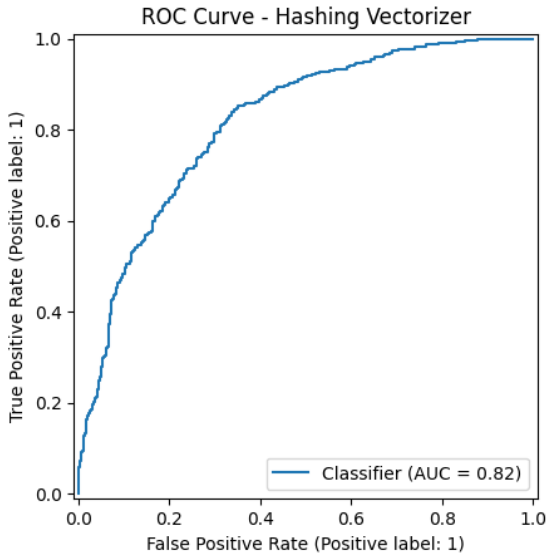


(a) ROC-AUC curve

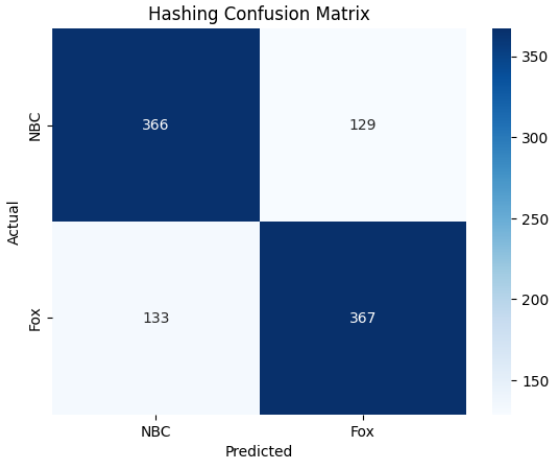


(b) Confusion Matrix

Figure 4: Naive Bayes with Count Vectorizer : ROC and Confusion Matrix



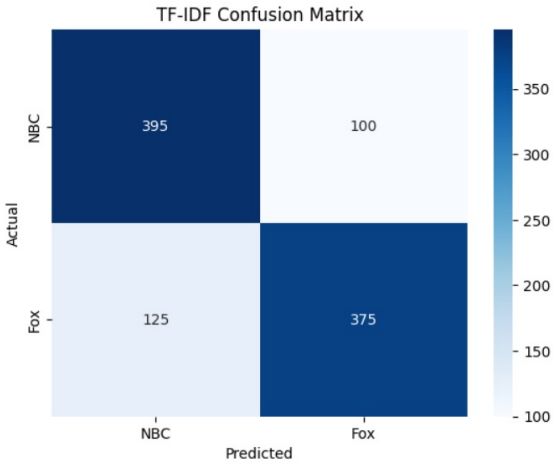
(a) ROC-AUC curve



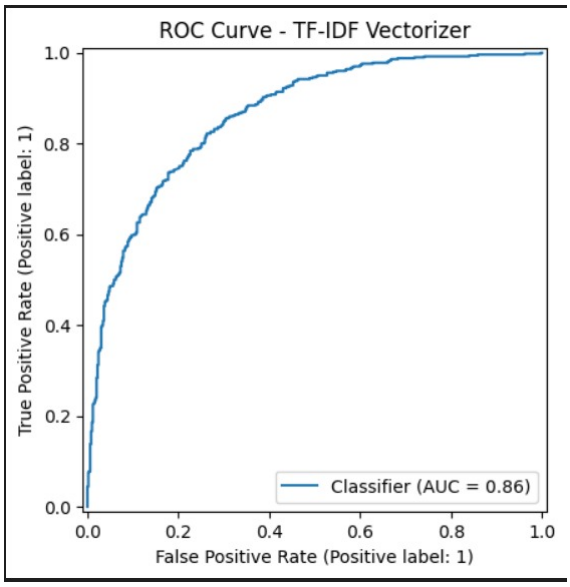
(b) Confusion Matrix

Figure 5: Naive Bayes with Hashing Vectorizer : ROC and Confusion Matrix

C. SVM

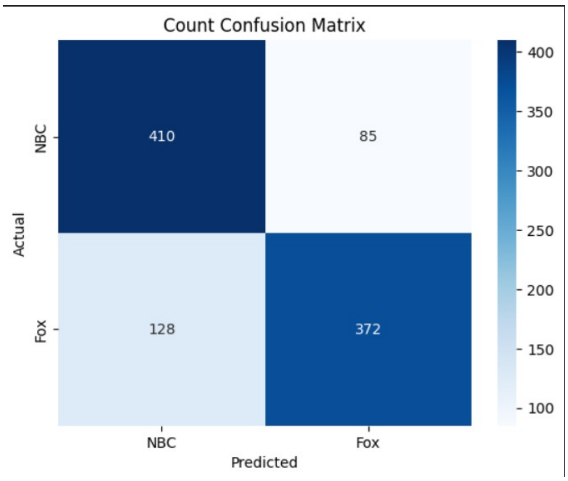


(a) confusion matrix

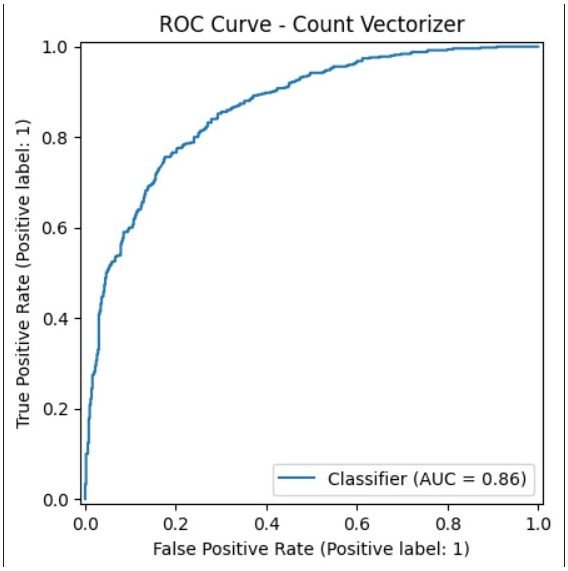


(b) Validation curve

Figure 6: SVM validation curve and confusion matrix

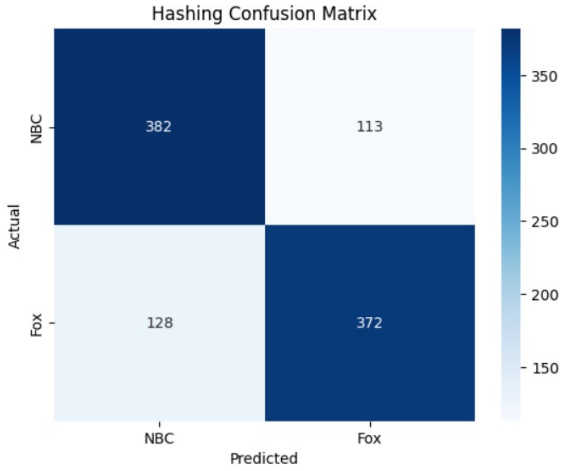


(a) confusion matrix

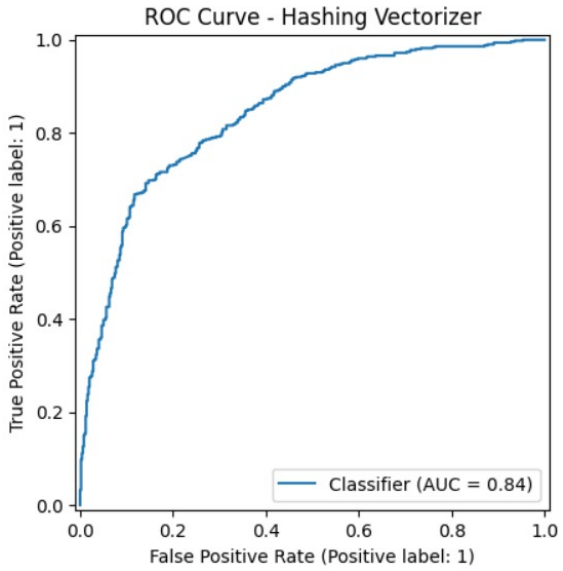


(b) Validation curve

Figure 7: SVM validation curves and confusion matrix



(a) Confusion matrix



(b) Validation curve

Figure 8: SVM validation curves and confusion matrix

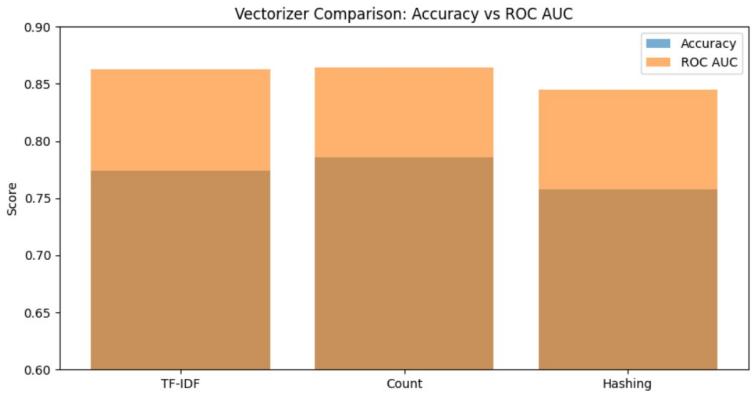
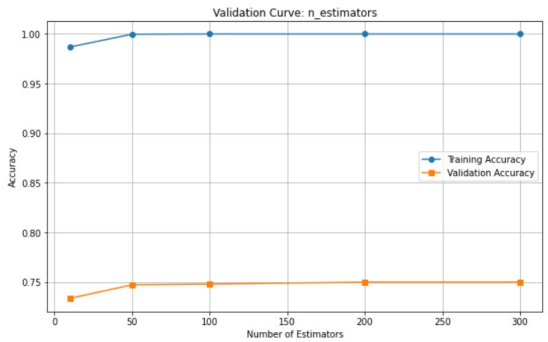
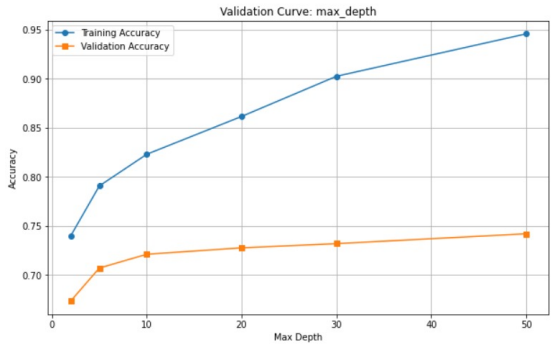


Figure 9: SVM ROC comparison

D. Random Forest

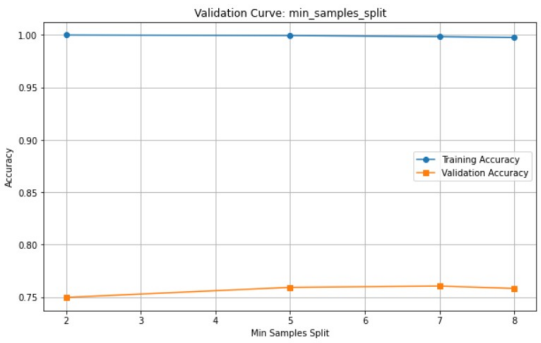


(a) Validation curve

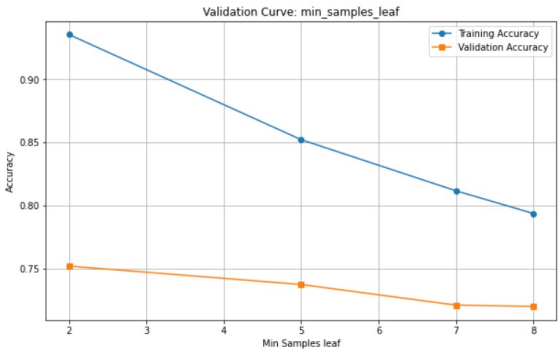


(b) Validation curve

Figure 10: Random Forest validation curves

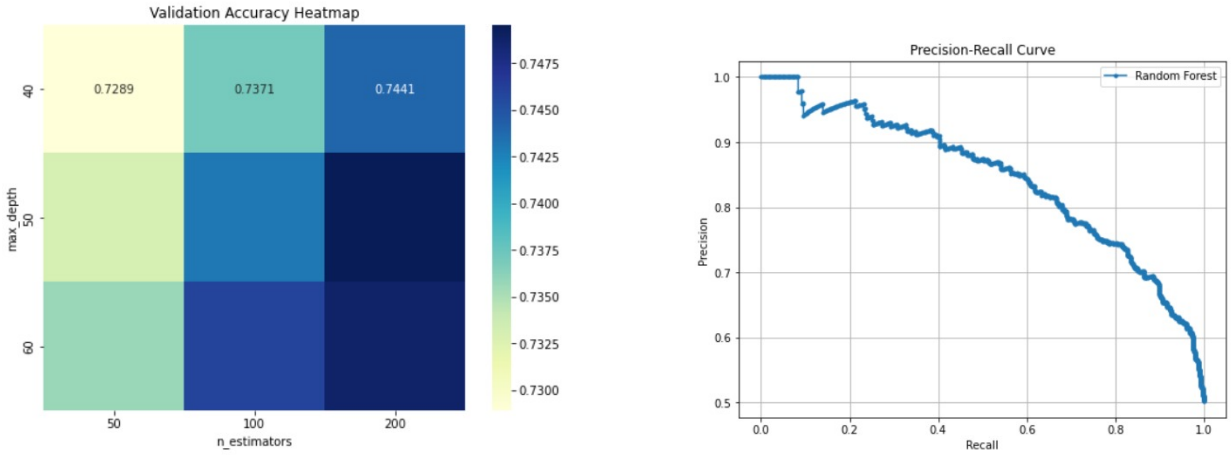


(a) Validation curve



(b) Validation curve

Figure 11: Random Forest validation curves



(a) Validation heatmap

(b) precision-recall curve

Figure 12: Random Forest validation curves

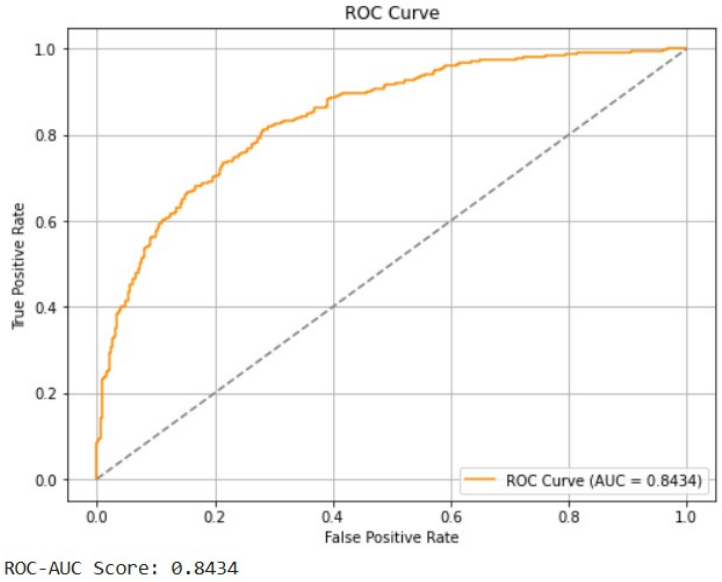
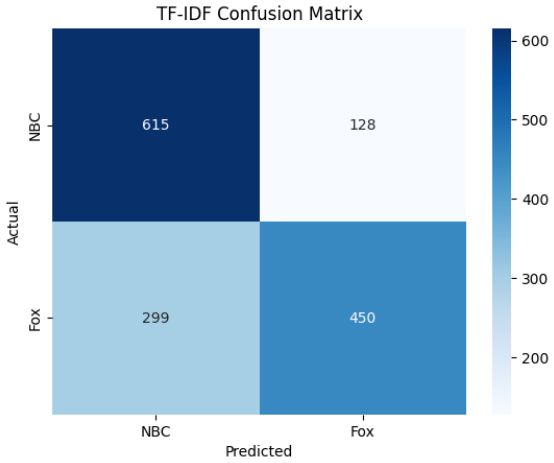
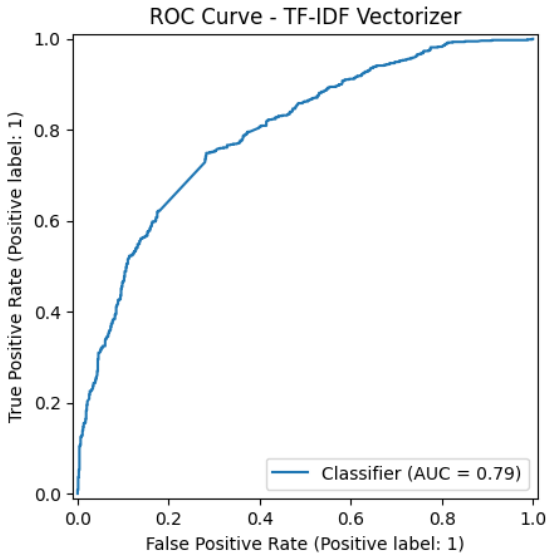


Figure 13: Random Forest ROC curve

E. XGBoost

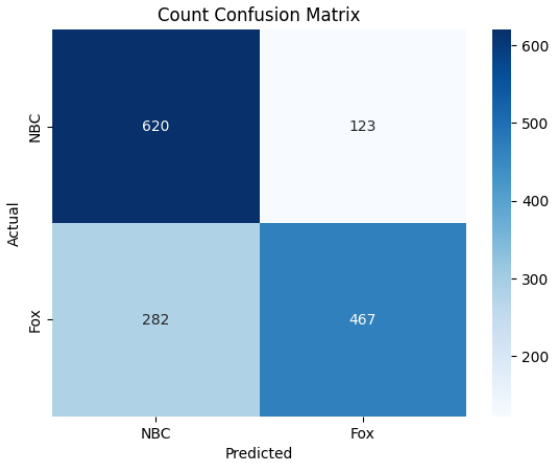


(a) confusion matrix

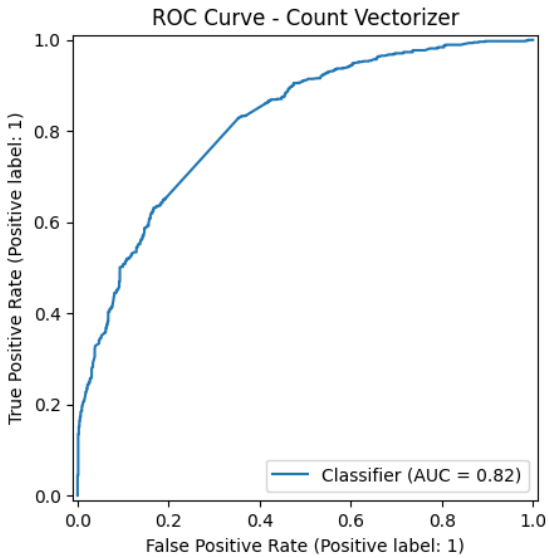


(b) Validation curve

Figure 14: XGB validation curve and confusion matrix

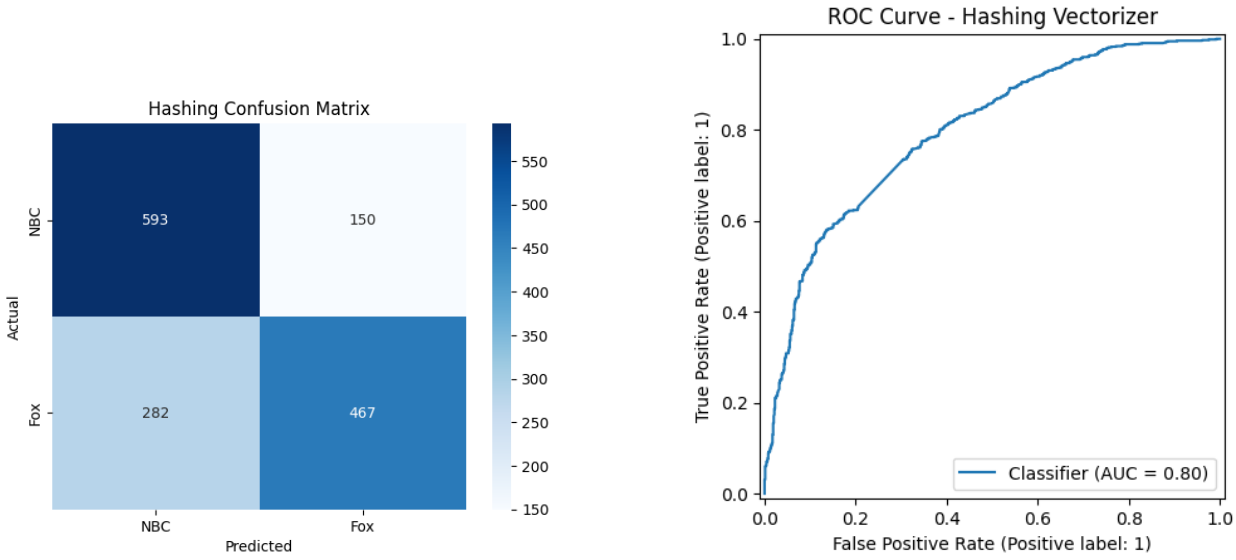


(a) confusion matrix



(b) Validation curve

Figure 15: XGB validation curves and confusion matrix



(a) Confusion matrix

(b) Validation curve

Figure 16: XGB validation curves and confusion matrix