**Fake News Detection Project Submission**

**1. Dataset Description**

For this project, we utilized the **GonzaloA/fake\_news** dataset from Hugging Face. This dataset contains labeled text data categorized as either **fake news (1)** or **real news (0)**. To enhance the robustness of our model, we considered additional datasets such as **LIAR** from Kaggle but ultimately focused on one source to maintain consistency.

**Dataset Statistics:**

* **Total Records**: 48,710
* **Training Set**: 38,972
* **Validation Set**: 8,117
* **Test Set**: 8,117
* **Features**:
  + text: The news article content.
  + label: Binary classification (1 = Fake, 0 = Real).

**2. Research Questions**

1. How effectively can different machine learning models classify fake vs. real news?
2. Which textual features contribute most to the detection of fake news?
3. How do traditional models (Random Forest, Decision Trees) compare with deep learning models (BERT-based Transformers)?
4. What are the trade-offs between interpretability and accuracy in different model types?

**3. Code Summary**

**Repository Link:**

GitHub Repository: <https://github.com/morb1212/final_project_learning_machine>

**Key Steps in the Code:**

1. **Data Preprocessing**:
   * Tokenization, removing punctuation, and converting text to lowercase.
   * Different vectorization methods: **TF-IDF, CountVectorizer, Word2Vec, FastText, Doc2Vec and BERT embeddings**.
2. **Model Training**:
   * Traditional ML Models: **Random Forest, Decision Trees, Logistic Regression, XGBoost, Gradient Boosting**.
   * Deep Learning: **BERT-based Transformer embeddings**.
3. **Evaluation Metrics**:
   * Accuracy, Precision, Recall, F1-score.
   * Confusion matrices for each model.
4. **Visualization & Analysis**:
   * Comparison of feature extraction methods using multiple embedding techniques.
   * Model comparison plots evaluating the impact of different embeddings.

**4. Summary of Algorithms & Performance**

**Performance Comparison Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **TF-IDF** | **CountVectorizer** | **Word2Vec** | **FastText** | **Doc2Vec** | **BERT** |
| **Random Forest** | **98.38%** | **98.36%** | **94.99%** | **94.76%** | **94.46%** | **94.72%** |
| **Decision Tree** | **97.33%** | **97.41%** | **89.24%** | **89.98%** | **80.58%** | **88.15%** |
| **Logistic Regression** | **96.67%** | **97.62%** | **96.45%** | **96.33%** | **95.24%** | **97.37%** |
| **XGBoost** | **98.38%** | **98.52%** | **96.33%** | **96.18%** | **95.32%** | **96.72%** |
| **Gradient Boosting** | **98.03%** | **98.03%** | **95.55%** | **95.46%** | **94.50%** | **95.94%** |

**Analysis of Performance**

* Best Performing Models: XGBoost and Random Forest achieved the highest accuracy (98.38% for TF-IDF, 98.52% for CountVectorizer).
* Decision Tree performed the worst among traditional models, especially with Doc2Vec (80.58%), indicating that linear-based embeddings may not work well with tree-based classifiers.
* Logistic Regression showed decent results, but it was consistently weaker than ensemble models like XGBoost and Gradient Boosting.
* BERT embeddings performed well, but the improvement over simpler methods like TF-IDF + XGBoost was marginal. Additionally, BERT was significantly more computationally expensive.

**Key Insights**

* TF-IDF and CountVectorizer were effective, achieving results close to deep learning methods but with much lower computational cost.
* Word2Vec, FastText, and Doc2Vec embeddings generally performed worse than TF-IDF, likely due to insufficient training on a small dataset compared to large pre-trained models.
* BERT embeddings were strong but not overwhelmingly superior to traditional models like XGBoost + TF-IDF, suggesting that simpler models might be preferable for real-time applications.
* If real-time inference is required, XGBoost with TF-IDF or CountVectorizer is the best choice, as it balances speed, accuracy, and interpretability.

**5. Challenges & How We Overcame Them**

**1. Slow Execution with BERT**

**Issue**: BERT-based embeddings took a long time to compute.

**Solution**:

* Implemented **batch processing** for BERT embeddings to reduce memory overload.
* Moved BERT computations to **GPU (NVIDIA RTX 2060)** for acceleration.

**2. Model Comparison & Feature Importance**

**Issue**: Understanding **why** models performed differently.

**Solution**:

* Found that **words like "breaking" and "exclusive"** were heavily associated with fake news.

**3.Impact of Different Text Vectorization Techniques**

**Issue**: Understanding how different feature extraction methods impact classification performance.

**Solution**:

* Implemented and compared multiple vectorization methods (TF-IDF, CountVectorizer, Word2Vec, FastText, Doc2Vec, BERT).
* Found that words like "breaking" and "exclusive" were heavily associated with fake news.

**4.Ensuring Reproducibility in Model Training**

**Issue:** In machine learning experiments, different runs can produce slightly different results due to randomness in dataset splits, model initialization, and optimization processes. This makes it difficult to compare models fairly.

**Solution:**

* Used **fixed random seeds** across all relevant functions to ensure that results remain consistent across different runs.

**5.** **Script Freezing Due to plt.show()**

**Issue:** The script got stuck after displaying confusion matrices.

**Solution:**

* Used plt.show(block=False) and plt.pause(3) to **prevent blocking execution.**

**•** Automatically saved each confusion matrix image.

**6. Final Insights & Conclusion**

* **Tree-based models (Random Forest & XGBoost) performed best**, combining high accuracy and interpretability.
* **Decision Tree was the weakest**, showing the importance of non-linear models for text classification.
* **Feature Engineering matters**: BERT embeddings performed well, but TF-IDF + XGBoost was much faster and nearly as accurate.
* **Deployment Considerations**: If real-time classification is required, **XGBoost with TF-IDF is preferable** due to speed.

**7. References & Further Work**

1. Hugging Face Dataset: GonzaloA/fake\_news
2. Kaggle LIAR Dataset: LIAR Fake News Dataset

**Future Work:**

* Fine-tune **BERT** on the dataset for even better accuracy.
* Experiment with **Ensemble Methods (Stacking Random Forest + BERT)**.
* Deploy the best-performing model using **Flask or FastAPI**.
* Investigating the impact of different hyperparameters on **Doc2Vec** and **FastText** models to improve their accuracy.

**Appendix: Confusion Matrices**

All confusion matrices have been saved as images and can be found in the project directory.

**Final Thought**: This project demonstrated how multiple machine learning approaches can effectively detect fake news, balancing accuracy, speed, and interpretability.

Comparison of Text Embedding Methods:

