**TEAM MEMBERS:**

* 200444008 Nazlıcan AYDIN
* 200444021 Kasım Berke ÜNVER
* 200444072 Muhammet Tarık ÖĞÜT

**News Article Categorization and Analysis Project**

**1. INTRODUCTION**

The goal of this project is to develop a Natural Language Processing (NLP)-based system that categorizes news articles into predefined categories such as Business, Entertainment, Politics, Sport, and Tech.  
We implement full text preprocessing, feature extraction using TF-IDF, and apply multiple machine learning models for classification and evaluation.

**2. DATA DESCRIPTION**

* **Dataset**: BBC News Dataset
* **Number of Articles**: 2225
* **Features**:
  + text: The full text content of each news article.
  + category: The corresponding category label. (Business, Entertainment, Politics, Sport, and Tech)

**3. IMPLEMENTATION STEPS**

**3.1 Preprocessing**

To ensure data quality and consistency, the following preprocessing steps were applied:

* **Named Entity Recognition (NER)**:  
  Leveraged spaCy'sen\_core\_web\_sm model to extract named entities (e.g., persons, locations, dates). These entities were later preserved during cleaning to retain meaningful context.
* def nameEntityRec(text):  
   doc= nlp(text)  
   entities=[(ent.text,ent.label\_) for ent in doc.ents]  
   return entities
* **Text Cleaning (Regex)**:
  + Removed URLs, mentions, hashtags, numbers, and special characters.
  + Removed single-character words.
  + Normalized whitespace.
* def textCleaning(text, preserved\_entities):  
   text=re.sub(r"http\S+|www\S+|https\S+",'',text)  
   text = re.sub(r'@\w+|#\w+', '', text)  
   text=re.sub(r'[^A-Za-z\s]','',text)  
   text= re.sub(r'\b\w\b', '',text)  
   text=re.sub(r'\s+',' ',text).strip()
* **Stopword Removal**:
  + Removed common English stopwords using **NLTK** to reduce noise.
* tokens=text.split()  
   tokens=[word for word in tokens if word.lower() not in stop\_words]  
   tokens+= [ent[0] for ent in preserved\_entities]  
   return ' '.join(tokens)  
    
  df\_train['Clean\_Text'] = df\_train.progress\_apply(  
   lambda row: textCleaning(row['text'], row['Entities']), axis=1)
* **Normalization (Lowercasing and Lemmatization)**:
  + Each token was lemmatized using **WordNetLemmatizer** and transformed to lowercase for normalization.
* def textNormalizing(text):  
   tokens=text.split()  
   lemmatized = [lemmatizer.lemmatize(token.lower()) for token in tokens]  
   return ' '.join(lemmatized)  
    
  df\_train['Lemmatized\_Text'] = df\_train['Clean\_Text'].apply(textNormalizing)
* **Typo Correction (Edit Distance)**:
  + A vocabulary of frequently occurring words was constructed. Then, **RapidFuzz's edit distance** method was applied to correct low-frequency misspellings.
* allWords = ' '.join(df\_train['text']).split()  
  wordFrequency= Counter(allWords)  
  vocabulary=set([word.lower() for word, frequency in wordFrequency.items() if frequency>2])  
    
  def correctTypo(word,vocab):  
   best\_match=process.extractOne(word,vocab,score\_cutoff=80)  
   return best\_match[0] if best\_match else word  
    
  def typoCorrection(text, vocab, preserved\_entities):  
   tokens = text.split()  
   corrected = [  
   token if token in preserved\_entities or token in vocab or len(token) <= 2 else correctTypo(token, vocab)  
   for token in tokens  
   ]  
   return ' '.join(corrected)  
    
  df\_train['Corrected\_Text'] = df\_train.progress\_apply(  
   lambda row: typoCorrection(row['Lemmatized\_Text'], vocabulary, row['Entities']), axis=1)

These preprocessing steps ensured that the final text corpus was clean, standardized, and semantically meaningful for classification.

**3.2 Feature Engineering**

* **Splitting Dataset into Train and Test Sets for Model Training**:
  + The dataset was divided using an 80-20 split, ensuring fair evaluation.
* X\_train, X\_test, Y\_train, Y\_test=train\_test\_split(  
   df\_train['Corrected\_Text'], df\_train['category'], test\_size=0.2, random\_state=42)
* **N-grams Extraction**:
  + Extracted both unigrams (single words) and bigrams (two-word phrases) using ngram\_range=(1,2).
* **TF-IDF Vectorization**:
  + Applied **TF-IDF (Term Frequency–Inverse Document Frequency)** to transform the text into numerical feature vectors.
  + Limited maximum features to **8000** to reduce dimensionality.
* vectorizer = TfidfVectorizer(ngram\_range=(1,3), max\_df=0.8, min\_df=3, max\_features=8000, sublinear\_tf=True, stop\_words='english')  
  X\_vec = vectorizer.fit\_transform(X)

**3.3 Classification Models**

Algorithms Used

* Logistic Regression
* Decision Tree
* Support Vector Machine (SVM)
* Multi-layer Perceptron (MLPClassifier)
* Naive Bayes
* Random Forest
* Voting Ensemble (LR + SVM + RF)
* models = {  
   "LogisticRegression": (LogisticRegression(max\_iter=1000), {'clf\_\_C': [0.1, 1, 10]}),  
   "DecisionTree": (DecisionTreeClassifier(random\_state=42), {'clf\_\_max\_depth': [10, 20]}),  
   "SVM": (LinearSVC(max\_iter=10000), {'clf\_\_C': [0.1, 1]}),  
   "MLP": (MLPClassifier(max\_iter=500), {'clf\_\_hidden\_layer\_sizes': [(100,)]}),  
   "NaiveBayes": (MultinomialNB(), {}),  
   "RandomForest": (RandomForestClassifier(), {'clf\_\_n\_estimators': [100]}),  
   "VotingEnsemble": (VotingClassifier(estimators=[  
   ('lr', LogisticRegression(max\_iter=1000)),  
   ('svm', LinearSVC(max\_iter=10000)),  
   ('rf', RandomForestClassifier())  
   ], voting='hard'), {})  
  }

**3.4 Evaluation Procedure**

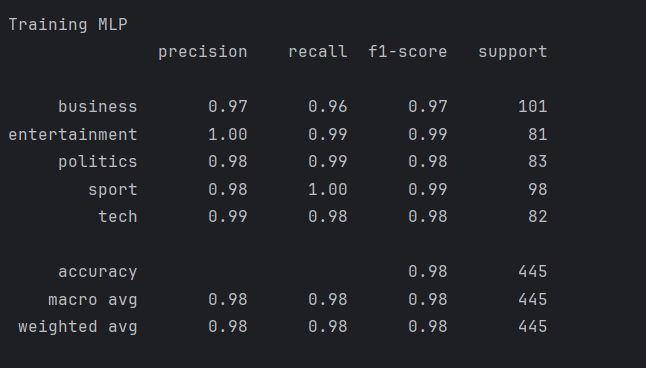
* GridSearchCV (on specified hyperparameters)
* 5-fold Stratified Cross Validation (f1\_macro)
* Accuracy, CV Score, Gap Analysis
* Confusion Matrix + Classification Report
* cv = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)  
  results = {}  
    
  for name, (model, param\_grid) in models.items():  
   print(f"Training {name}")  
   pipe = Pipeline([('clf', model)])  
   grid = GridSearchCV(pipe, param\_grid, cv=cv, scoring='f1\_macro', n\_jobs=-1)  
   grid.fit(X\_train, Y\_train)  
    
   best\_model = grid.best\_estimator\_  
   Y\_pred = best\_model.predict(X\_test)  
    
   acc\_train = accuracy\_score(Y\_train, best\_model.predict(X\_train))  
   acc\_test = accuracy\_score(Y\_test, Y\_pred)  
   cv\_scores = cross\_val\_score(best\_model, X\_train, Y\_train, cv=cv, scoring='f1\_macro')  
    
   results[name] = {  
   'model': best\_model,  
   'train\_acc': acc\_train,  
   'test\_acc': acc\_test,  
   'cv\_score': cv\_scores.mean(),  
   'cv\_std': cv\_scores.std(),  
   'params': grid.best\_params\_  
   }  
    
   print(classification\_report(Y\_test, Y\_pred))  
    
  best\_model\_name = max(results, key=lambda k: results[k]['test\_acc'])  
  joblib.dump(results[best\_model\_name]['model'], "news\_classifier.pkl")  
  joblib.dump(vectorizer, "tfidf\_vectorizer.pkl")  
    
  best\_model = results[best\_model\_name]['model']  
  y\_pred\_best = best\_model.predict(X\_test)  
  labels = sorted(y.unique())  
    
  cm = confusion\_matrix(Y\_test, y\_pred\_best, labels=labels)  
    
  plt.figure(figsize=(8, 6))  
  sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)  
  plt.xlabel("Predicted Label")  
  plt.ylabel("True Label")  
  plt.title(f"Confusion Matrix - Best Model ({best\_model\_name})")  
  plt.tight\_layout()  
  plt.show()

**4. Best Model:** MLPClassifier

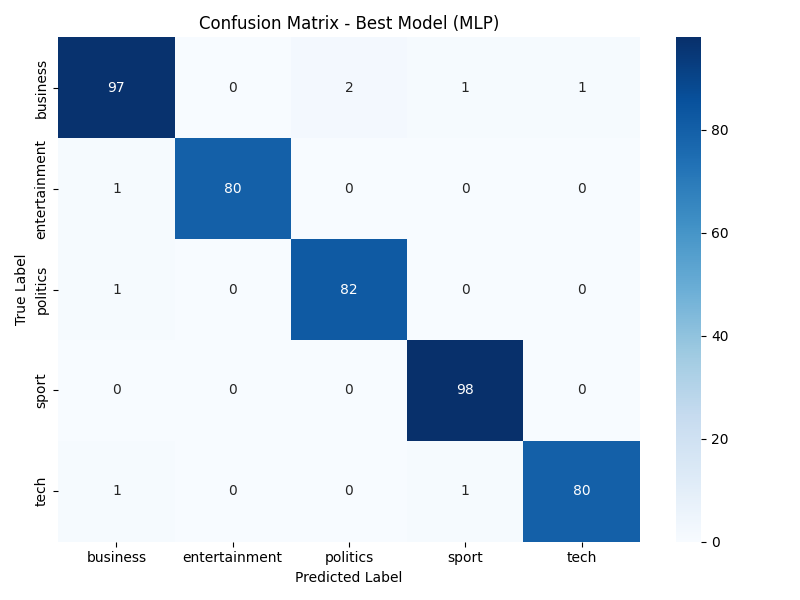
****

**4.1 Evaluation Metrics**

* **Test Accuracy**: 0.9820
* **Macro F1 Score**: 0.98
* **Best Performing Class**: Sport (F1 = 0.99)
* **Lowest Class Performance**: Politics (F1 = 0.98)
* **Cross-validation score**: Consistently above 0.97 across folds.



**4.2 Confusion Matrix**



The confusion matrix above illustrates the performance of the best-performing model, MLPClassifier, on the test set. Each row corresponds to the actual class label, while each column represents the predicted class by the model.

* **Overall Accuracy**: The classifier demonstrates **high precision across all classes**. Notably, the sport category is classified with almost perfect accuracy — all 98 instances are predicted correctly.
* **Business**: Out of 101 instances, 97 were correctly predicted. Misclassifications include 2 labeled as politics, 1 as sport, and 1 as tech, resulting in a 96% accuracy for this class.
* **Entertainment**: Only 1 instance was misclassified (business), yielding a near-perfect classification rate of ~99%.
* **Politics**: 82 out of 83 instances were accurately identified, with a single misclassification (business).
* **Tech**: Two misclassifications occurred — one as business, and another as sport, leading to an accuracy of approximately 97.5%.

There is no strong evidence of systematic confusion between any specific pair of classes. The misclassifications are minor and scattered, indicating the model generalizes well rather than overfitting. The classifier maintains balanced performance across all five categories, which is essential in multi-class classification tasks.

**5. Model Comparison**

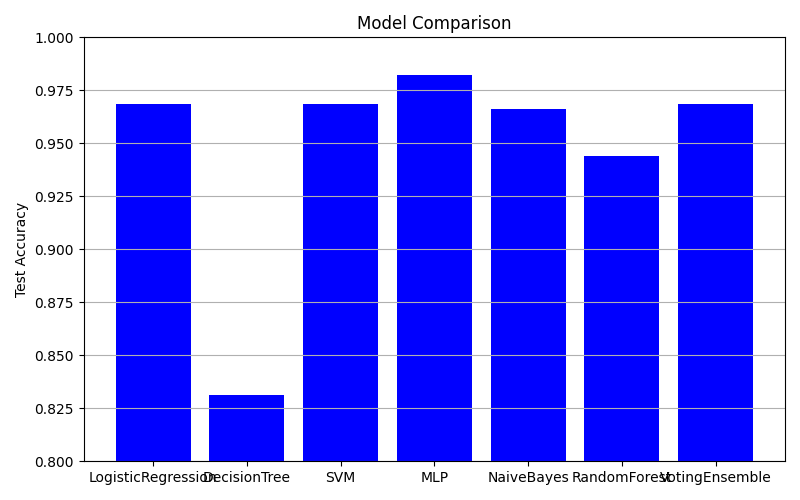
In order to evaluate different machine learning algorithms for news article classification, seven distinct models were trained and compared: Logistic Regression, Decision Tree, Support Vector Machine (SVM), Multi-Layer Perceptron (MLP), Naive Bayes, Random Forest, and a Voting Ensemble.Each model was assessed using the same TF-IDF feature representation and evaluated on a 20% test split of the BBC news dataset.

**Performance Overview:**

| **Model** | **Accuracy (%)** |
| --- | --- |
| **MLPClassifier** | **98.20** |
| Logistic Regression | 97.00 |
| SVM | 97.00 |
| Naive Bayes | 97.00 |
| Voting Ensemble | 97.00 |
| Random Forest | 94.00 |
| Decision Tree | 83.00 |

* MLPClassifier achieved the highest accuracy (98.20%), making it the best-performing model in this task. It consistently showed high precision, recall, and F1-scores across all categories.
* Logistic Regression, SVM, Naive Bayes, and the Voting Ensemble models followed closely, each with an accuracy of 97%, demonstrating strong and balanced performance.
* Decision Tree underperformed relative to the others, with only 83% accuracy, likely due to overfitting and limited generalization capability on text-based features.
* Random Forest performed better than Decision Tree but still fell short of the top models, with a test accuracy of 94%.

The results clearly show that deep models like **MLPClassifier** are capable of capturing non-linear relationships in text classification tasks. While simpler models such as Naive Bayes and Logistic Regression also perform well, the MLP's superior performance justifies its selection as the **final model** for deployment and interpretation tasks.

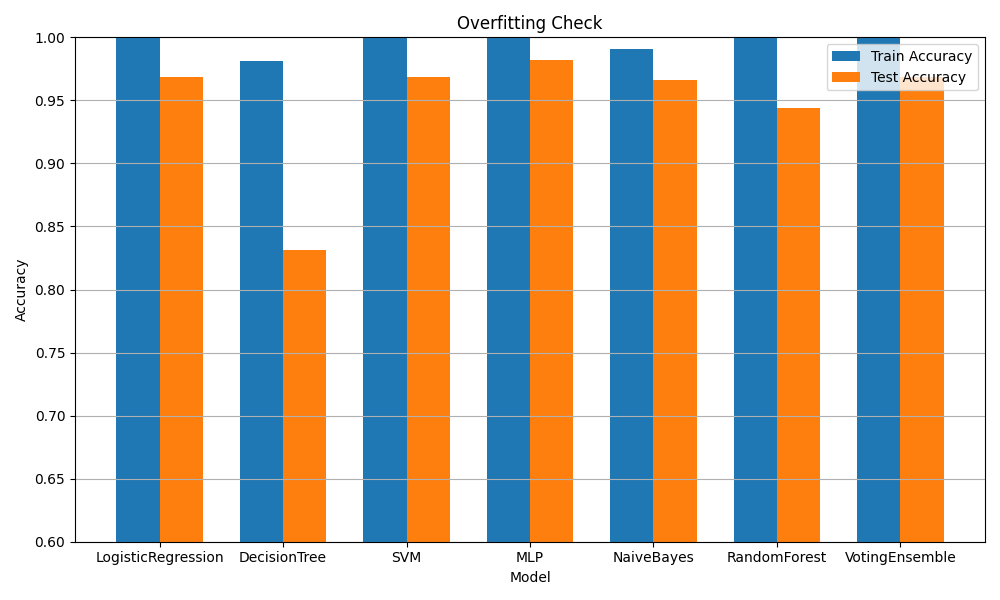


**6.Overfitting Check**

In order to assess the generalization capabilities of each model, both training and test accuracies were compared. A significant gap between these two scores can indicate overfitting, where the model performs well on training data but fails to generalize to unseen examples.

* MLPClassifier, which was identified as the best model, exhibited excellent generalization, with train and test accuracies both around 98%. This suggests that the model is neither underfitting nor overfitting.
* Logistic Regression, SVM, and Naive Bayes also demonstrated balanced performance with minimal train-test accuracy gaps.
* Decision Tree displayed the widest gap, with a high training accuracy (~98%) and significantly lower test accuracy (~83%). This clearly indicates overfitting due to its tendency to memorize the training data.
* Random Forest improved upon this but still showed a noticeable gap, likely due to its ensemble nature compensating for individual overfitted trees.
* Voting Ensemble yielded stable and consistent results, closely matching those of the top individual classifiers.

Models like MLP, SVM, and Logistic Regression not only performed well but also generalized reliably, confirming their robustness. On the other hand, Decision Tree was prone to overfitting and may not be suitable without pruning or additional regularization techniques.

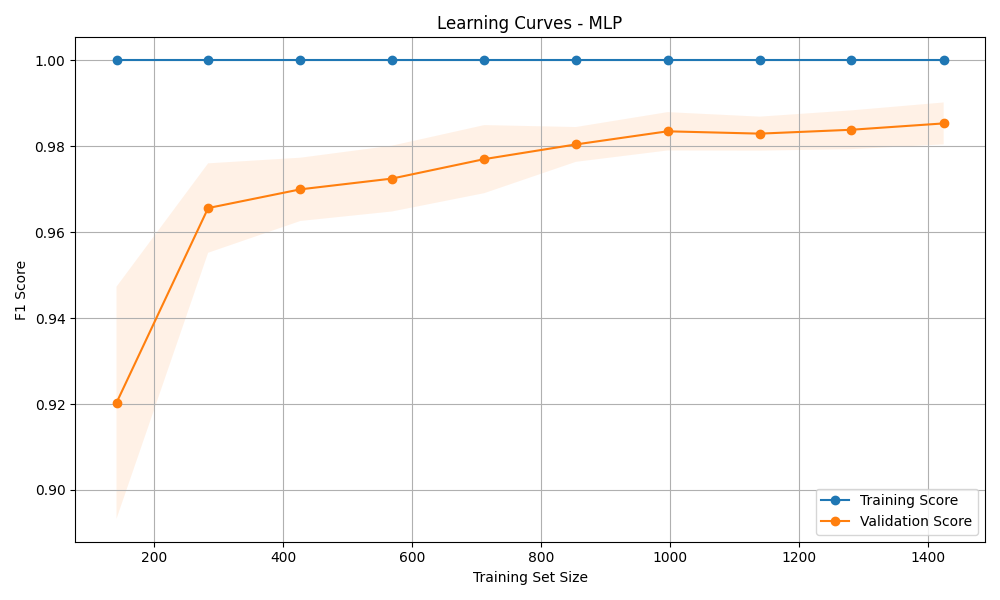


**7.Learning Curves Analysis – MLP Classifier**

Learning curves illustrate how a model's performance changes with increasing training data size. They help determine whether the model suffers from high bias (underfitting) or high variance (overfitting).

* The blue line (training F1-score) remains nearly constant and close to 1.0, indicating that the MLPClassifier learns the training data exceptionally well.
* The orange line (validation F1-score) starts slightly lower but steadily improves as more data is used for training, eventually stabilizing around 0.985.
* The gap between training and validation curves is small, which confirms that the model generalizes well without overfitting.
* The shaded region around the validation curve indicates low variance across cross-validation folds, reinforcing the model’s stability.

The MLPClassifier benefits from more training data and maintains consistent generalization. The curves demonstrate that the model is well-optimized and has high capacity without overfitting, making it an excellent choice for this news categorization task.



**8.LIME Analysis – Model Interpretability**

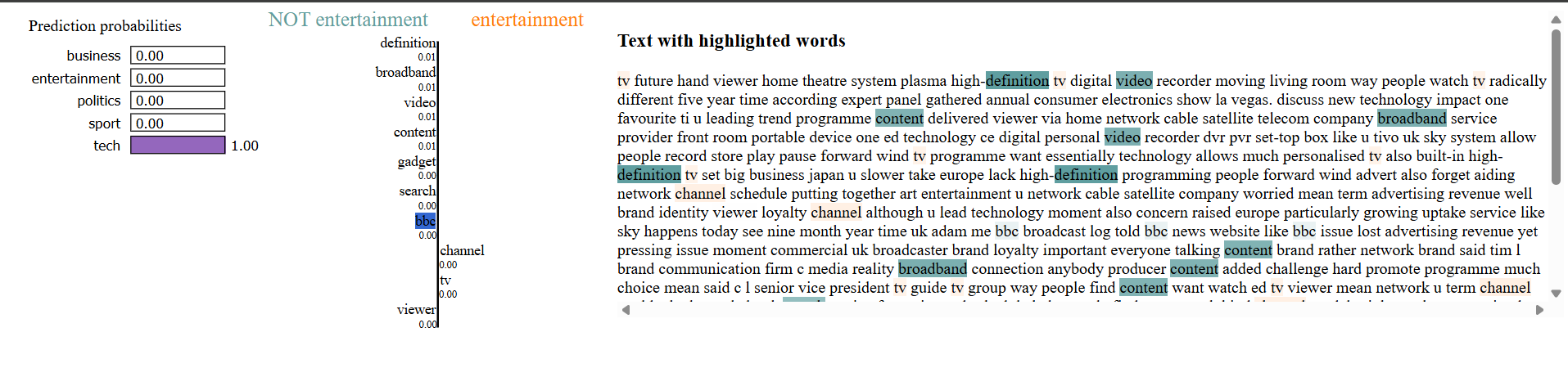
LIME (Local Interpretable Model-agnostic Explanations) is an algorithm designed to explain the predictions of any machine learning model in an interpretable and human-understandable way. It works by approximating the model locally with an interpretable one (e.g., linear model) around the prediction of interest.

LIME provides insight into why a model made a certain prediction by highlighting the words or features that contributed the most to the predicted class.

In our project, the best performing model was MLPClassifier, which is a black-box neural network. While accurate, neural networks do not inherently explain why they make decisions, which can make debugging and trust-building difficult.

To overcome this challenge, we used LIME to:

* Visualize which words influenced the model's decision.
* Increase transparency and interpretability.
* Confirm whether the model is relying on relevant and meaningful features (words) or biased/noisy tokens.



* The given example was confidently classified as tech with a prediction probability of 1.00.
* Words such as definition, video, content, broadband, and tv were highlighted as positive contributors to the tech label.
* This confirms that the model is making decisions based on semantically appropriate keywords that are strongly related to technology.

LIME added model interpretability to our high-performance MLP model. It helped ensure that the classifier was not only accurate but also explainable, making the system more robust and trustworthy—especially critical in real-world applications involving automated text classification.

**9.Streamlit Web Interface**

To make our news article classifier interactive and user-friendly, we developed a lightweight web interface using Streamlit, a popular Python library for building data apps.

The Streamlit interface enables users to:

* Paste or type a news article
* Classify the article in real-time into one of five categories: business, entertainment, politics, sport, or tech
* View the predicted category and class probabilities

**Features:**

* **Title and Description Section**: Introduces the app and its purpose.
* **Text Area**: Allows users to input any arbitrary news text.
* **Prediction Button**: When clicked, processes the input through the trained MLPClassifier model.
* **Predicted Category**: Clearly displayed in a highlighted section.
* **Class Probabilities**: Shows confidence levels across all possible categories.
* **Model Tag**: Indicates which model was used for classification (MLPClassifier).

By combining Streamlit with our machine learning pipeline, we successfully transformed a backend classifier into a fully functional interactive tool. This enhances accessibility and serves as a valuable showcase for real-time NLP applications.

