ChatGPT

Fake News Detection with TF-IDF and Logistic Regression

Introduction

Fake news detection is the task of classifying news articles as **real (true)** or **fake (false)**. In this project, we use a classic natural language processing approach: represent text using **TF-IDF features** and train a **logistic regression** classifier. This approach is explainable and grounded in fundamental mathematics, making it suitable for an AI Engineering course with a focus on classical NLP techniques (no deep learning). Below, we provide the mathematical foundations (with LaTeX-formatted equations) for TF, IDF, TF-IDF, and logistic regression, followed by a step-by-step Jupyter Notebook example using a small fake news dataset. All code is in Python with clear comments for beginners.

Mathematical Foundations

Term Frequency (TF)

Term Frequency **TF** measures how often a term (word) appears in a document, relative to the document's length. If $f_{t,d}$ is the raw count of term t in document d (i.e. the number of times t occurs in d), one common definition of TF is:

$$\mathrm{TF}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}\,.$$

This formula means we divide the count of term *t* by the total number of terms in document *d*, yielding a normalized frequency [1]. (In practice, sometimes simpler versions are used, such as $TF(t,d) = f_{t,d}$ without normalization [2].)

Inverse Document Frequency (IDF)

Inverse Document Frequency **IDF** measures how rare a term is across the entire corpus of documents. Let $N = |D|$ be the total number of documents in the corpus D, and let $df_t = |\{d \in D : t \in d\}|$ be the number of documents that contain term *t*. The IDF for term *t* is typically defined as:

$$\mathrm{IDF}(t,D) = \log \frac{N}{df_t}\,.$$

In words, IDF is the logarithm of the total number of documents divided by the number of documents containing *t*. Rare terms (low df_t) get a higher IDF, while common terms (high df_t) get a lower IDF [3]. (Often a smoothing term is added to avoid division by zero [4], but the above is the core idea.)

TF-IDF Weighting

The **TF-IDF** score of a term is simply the product of its TF and IDF values:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)\,.$$

This weighting scheme gives a high score to terms that are frequent in a given document *d* (high TF) but rare in the corpus *D* (high IDF) [5]. Conversely, terms that are too common across documents (like stopwords "the", "and", etc.) get down-weighted by a low IDF. Thus, TF-IDF filters out ubiquitous words and highlights more informative, distinguishing terms in each document [5].

Logistic Regression Model

Logistic regression is a **binary classification** model that outputs a probability between 0 and 1 for the positive class (e.g., "fake news"). It uses the **sigmoid (logistic) function** to map a linear combination of features to a probability. For feature vector $\mathbf{x}$ and weight vector $\mathbf{w}$ (with bias b), the model's prediction (the probability that the label $y = 1$, i.e. "fake") is:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}, \qquad \text{where } z = \mathbf{w} \cdot \mathbf{x} + b.$$

Here $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function [6]. This S-shaped function outputs values in the range (0,1), so if $\hat{y} \approx 0.9$ it means the model is 90% confident the news is fake, whereas $\hat{y} \approx 0.1$ would indicate a low probability of being fake.

The logistic sigmoid function $\sigma(z)$ plotted in blue. It maps any real-valued input z to a value between 0 and 1, making it useful for modeling probabilities. The curve is steepest at $z = 0$ (output 0.5), and flattens out towards 0 and 1 for extreme inputs. Logistic regression uses this curve to translate a linear score $z = \mathbf{w} \cdot \mathbf{x} + b$ into a probability of the positive class.

Likelihood and Learning: Given a dataset of examples, logistic regression learns the weights $\mathbf{w}, b$ by **maximum likelihood estimation**. If $y \in \{0, 1\}$ are the true labels and $\hat{y}_i = \sigma(\mathbf{w} \cdot \mathbf{x}_i + b)$ are the predicted probabilities, the likelihood of the observed data (assuming independent samples) is:

$$L(\mathbf{w}) = \prod_{i=1}^{N} [\hat{y}_i]^{\,y_i} \, [1 - \hat{y}_i]^{\,1-y_i}\,.$$

Taking the log of this likelihood gives the **log-likelihood** (which is easier to maximize):

$$\ell(\mathbf{w}) = \sum_{i=1}^{N} \Big(y_i \ln \hat{y}_i \;+\; (1 - y_i) \ln(1 - \hat{y}_i) \Big)\,.$$

Maximizing $\ell(\mathbf{w})$ yields the best-fit parameters [7] [8]. There is no closed-form solution for $\mathbf{w}$ in logistic regression, so one must use **iterative optimization**. **Gradient descent** is a common method: we compute the gradient of the log-likelihood with respect to the weights and update $\mathbf{w}$ in the direction that increases ℓ. Intuitively, if a term w_j multiplies a feature x_j that is positively correlated with the article being fake, the optimization will adjust w_j to be larger, increasing the predicted probability for fake news. Modern libraries

like scikit-learn handle this optimization for us (using techniques like gradient descent or related algorithms under the hood).

Implementation Example (TF-IDF + Logistic Regression)

Below is a walkthrough of a simple fake news detection model implemented in a Python Jupyter Notebook. We use a small dataset of news articles with binary labels indicating whether each news piece is fake or real. (For example, the **Kaggle Fake News** dataset contains news articles with columns for id, title, author, text, and label [9].) We will:

1. **Load and preprocess the data** – read the dataset, handle missing values, and prepare text for modeling.
2. **Vectorize text using TF-IDF** – convert the news articles into numerical feature vectors.
3. **Train a logistic regression classifier** on the TF-IDF features.
4. **Evaluate** the model using accuracy and a confusion matrix.

Data Loading and Preprocessing

First, we load the data using pandas. In our dataset (e.g. from Kaggle), each entry has the following fields [9]:

- `id`: Unique identifier for the news article
- `title`: Headline of the article
- `author`: Author of the article
- `text`: Full text of the article (could be incomplete)
- `label`: Ground truth class (1 = fake/unreliable, 0 = real news)

For simplicity, we will combine the title and text into a single text field for each article, since both can provide useful cues. We also need to handle any missing values (e.g. if an article text or author is missing, we can fill it with an empty string). Here's the code:

```
import pandas as pd

# Load the dataset (assumes a CSV file with columns: id, title, author, text,
label)
data = pd.read_csv("train.csv")  # replace with the actual path to the dataset
print(f"Loaded {data.shape[0]} articles with {data.shape[1]} columns.")

# Fill missing text or title with empty strings (in case there are NaNs)
data['title'] = data['title'].fillna('')
data['text']  = data['text'].fillna('')

# Combine title and text into one field (optional but often useful)
data['content'] = data['title'] + " " + data['text']
```

```
# Look at a sample of the data
print(data[['title','author','label']].head(3))
```

Explanation: We use `pandas.read_csv` to load the dataset into a DataFrame. We fill any missing `title` or `text` with `''` (empty string) to avoid issues. We then create a new column `content` by concatenating the title and text, so that our feature will be the entire content of the article. Finally, we print the first few entries to verify the data loaded correctly.

TF-IDF Vectorization

Next, we convert the text content into TF-IDF feature vectors using scikit-learn's `TfidfVectorizer`. This will tokenize the text, count term frequencies, and scale by inverse document frequency. We can also configure the vectorizer to ignore very common English stopwords (like "the", "and") which are not informative for classification. We might limit the number of features to a reasonable size (e.g., 5000 or 10000 most frequent terms) to keep the model tractable for a small dataset. Here's how to do it:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize TF-IDF vectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
# max_features=5000 limits the vocabulary to 5000 most important terms, for
speed.

# Learn vocabulary from the content and transform content to TF-IDF matrix
X = vectorizer.fit_transform(data['content'])

# Extract the target labels
y = data['label'].values

print(f"TF-IDF feature matrix size: {X.shape}")
```

Explanation: We create a `TfidfVectorizer` with English stopwords removed. Calling `fit_transform` on the text data does two things: it learns the vocabulary of terms in the corpus and computes the TF-IDF weights for each document. The result `X` is a matrix of size (number_of_articles, number_of_features). Each row corresponds to an article, and each column corresponds to a specific term's TF-IDF weight. The print statement might output something like `TF-IDF feature matrix size: (20000, 5000)` meaning 20,000 articles and 5,000 features (if our dataset had 20k articles). Each feature is a weighted word count.

Training the Logistic Regression Model

With our features `X` and labels `y`, we can train a logistic regression classifier. We will split the data into a training set and a test set to evaluate performance on unseen data. A typical split is 80% for training and 20% for testing. We use `sklearn.model_selection.train_test_split` for this. Then we create a `LogisticRegression` model and fit it to the training data.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Split data into training and testing sets (e.g., 80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize Logistic Regression model
model = LogisticRegression(max_iter=1000)  # max_iter increased to ensure
convergence

# Train the model on the training data
model.fit(X_train, y_train)

print("Model training complete.")
```

Explanation: We set aside 20% of the data for testing, using a fixed `random_state` for reproducibility. We instantiate `LogisticRegression` from scikit-learn. (By default, it uses an optimization solver to find the best weights $\mathbf{w}, b$ that maximize the log-likelihood as discussed earlier.) We set `max_iter=1000` to give the solver more iterations to converge, since TF-IDF data can be high-dimensional. The `model.fit()` call carries out the training — behind the scenes, this uses techniques like gradient descent or coordinate descent to optimize the weights. After fitting, the model has learned coefficients for each TF-IDF feature.

Model Evaluation (Accuracy and Confusion Matrix)

Now we evaluate the trained model on the test set. We'll use **accuracy** as a basic metric, which is the fraction of news articles correctly classified. We will also compute the **confusion matrix** to see the breakdown of true vs. predicted labels (how many real news and fake news were correctly vs. incorrectly classified).

```
from sklearn.metrics import accuracy_score, confusion_matrix

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate accuracy
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy on test set: {acc:.2%}")

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:\n", cm)
```

Explanation: The `model.predict` method applies the logistic regression model to each test article's TF-IDF vector, outputting a predicted label (0 or 1). We compare these predictions `y_pred` to the true

labels `y_test`. The `accuracy_score` gives the proportion of correct predictions. The confusion matrix `cm` is a 2x2 array: `cm[0,0]` is the count of real news correctly identified as real (true negatives if we consider "fake" as the positive class), `cm[1,1]` is the count of fake news correctly identified as fake (true positives), and the off-diagonals `cm[0,1]`, `cm[1,0]` are the misclassified counts (false positive and false negative respectively).

For instance, an output might be:

```
Accuracy on test set: 94.50%
Confusion matrix:
 [[1900,  120],
  [ 130, 1950]]
```

This indicates out of 2020 real news articles, 1900 were correctly flagged as real and 120 were mistakenly marked fake; out of 2080 fake news articles, 1950 were correctly flagged as fake and 130 were missed. **Accuracy** in this hypothetical scenario is roughly 94.5%, which is quite good for a simple model.

You can also derive other metrics from the confusion matrix, such as precision, recall, or F1-score, for a more detailed evaluation. For example, a high false positive count (mislabeling real news as fake) might be particularly undesirable, so we might examine **precision** for the "fake" class, etc. In this basic project, however, accuracy and the confusion matrix provide a sufficient starting point.

Conclusion

In this project, we demonstrated a complete fake news detection pipeline using TF-IDF features and logistic regression. We covered the mathematical understanding of TF, IDF, TF-IDF, and the workings of logistic regression (sigmoid function, likelihood, and how learning is achieved via optimization). The Python implementation showed how to go from raw text data to a trained classification model using scikit-learn, with each step explained. This approach is **interpretable and grounded in fundamental concepts**: for example, the model's weights on TF-IDF features can indicate which words are most indicative of fake vs. real news. While more advanced techniques (like neural networks or transformers) can achieve higher accuracy, this classical method remains a valuable baseline and teaching tool in an AI Engineering context. By completing this project, students gain practical experience with text preprocessing, feature extraction, and model evaluation, all while reinforcing their understanding of the underlying mathematics of NLP and logistic regression.

---

1 3 TF-IDF — Term Frequency-Inverse Document Frequency – LearnDataSci
https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/

2 4 5 tf–idf - Wikipedia
https://en.wikipedia.org/wiki/Tf%E2%80%93idf

6 Logistic Regression using the sigmoid function - Supervised ML: Regression and Classification - DeepLearning.AI
https://community.deeplearning.ai/t/logistic-regression-using-the-sigmoid-function/762089

7 8 Logistic regression - Wikipedia
https://en.wikipedia.org/wiki/Logistic_regression

9 Fake News Detection in Python - The Python Code
https://thepythoncode.com/article/fake-news-classification-in-python