NLP-Based Classification of Haitian Disaster Response Messages

Table of Contents

1. Project summary

2. Business understanding

In [1]:

```
# core libraries
import pandas as pd
import numpy as np
import re
import string
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud

# nlp tools
import nltk
import ssl
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

# sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (accuracy_score, precision_recall_fscore_support,
                             classification_report, confusion_matrix, roc_auc_score)
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score, f1_score, precision_recall_fscore_support, cl
assification_report, confusion_matrix


# suppress warnings
warnings.filterwarnings('ignore')

# download NLTK resources
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('omw-1.4', quiet=True)
```

Out[1]:

```
True
```

3. Data collection and description

The analysis uses two structured CSV files from the `Disaster Response Messages` dataset, originally published by Figure Eight (now Appen). This dataset supports humanitarian AI development through real-world crisis communications.

`disaster_messages.csv` **contains raw message records with four key fields:**

- **id: A unique identifier for each message**
- **message: An English-translated version of the original text, used as the primary input for modeling**
- **original: The message in its native language, typically Haitian Creole or French**
- **genre: The source channel, categorized as direct (firsthand pleas), news (third-party reports), or social (public posts, often from Twitter)**

`disaster_categories.csv` **provides multi-label annotations for each message in a single semicolon-delimited string ("request-1;offer-0;food-1;water-1;..."). Each segment encodes a humanitarian category and its binary label (1 = present, 0 = absent).**

In [2]:

```
# load messages dataset
messages = pd.read_csv("disaster_messages.csv")
print("Messages columns:", messages.columns.tolist())
```

```
Messages columns: ['id', 'message', 'original', 'genre']
```

In [3]:

```
# load categories dataset
cat = pd.read_csv("disaster_categories.csv")
print("Categories columns:", cat.columns.tolist())
```

```
Categories columns: ['id', 'categories']
```

The two files share a common id column, which enables reliable record linkage.

3.1 Data source context and composition

The dataset originates from crowdsourced annotation efforts during three major disasters: the 2010 Haiti earthquake, Hurricane Sandy (2012), and the Nepal earthquake (2015). Messages were gathered from SMS, social media, and news outlets, then translated into English and labeled across 36 humanitarian categories by multiple human annotators to ensure consistency.

For this project, the scope is limited to Haiti-related direct messages only, as the operational goal is to support response efforts specific to the 2010 earthquake. This focus ensures contextual relevance and avoids dilution from unrelated global events.

In [4]:

```
print("Messages shape:", messages.shape)
print("Categories shape:", cat.shape)
```

```
Messages shape: (26248, 4)
Categories shape: (26248, 2)
```

The raw data consists of 26 382 message records, with matching entries in both files. All messages include

complete id, message, and genre fields, and every record has a corresponding category annotation.

3.2 Merging datasets

In [5]:

```
# merge the datasets
df = messages.merge(cat, left_on='id', right_on='id', how='inner')
df.head()
```

Out[5]:

	id	message	original	genre	categories
0	2	Weather update - a cold front from Cuba that c...	Un front froid se retrouve sur Cuba ce matin. ...	direct	related-1;request-0;offer-0;aid_related-0;medi...
1	7	Is the Hurricane over or is it not over	Cyclone nan fini osinon li pa fini	direct	related-1;request-0;offer-0;aid_related-1;medi...
2	8	Looking for someone but no name	Patnm, di Maryani relem pou li banm nouvel li ...	direct	related-1;request-0;offer-0;aid_related-0;medi...
3	9	UN reports Leogane 80-90 destroyed. Only Hospi...	UN reports Leogane 80-90 destroyed. Only Hospi...	direct	related-1;request-1;offer-0;aid_related-1;medi...
4	12	says: west side of Haiti, rest of the country ...	facade ouest d Haiti et le reste du pays aujou...	direct	related-1;request-0;offer-0;aid_related-0;medi...

In [6]:

```
df.shape
```

Out[6]:

```
(26386, 5)
```

The two files are joined on the id column using an inner join to create a unified dataset. This merge produces a single table with 26 382 rows and 5 initial columns (id, message, original, genre, categories). The resulting structure serves as the foundation for all downstream processing, filtering, and modeling.

4. Data understanding and preparation

4.1 Initial Inspection

We inspect the first few rows and assess data quality:

In [7]:

```
# missing values
print("\nMissing values in dataset:\n", df.isnull().sum())

# duplicates
print(f"\nDuplicate message IDs: {df.duplicated(subset='id').sum()}")
```

```
Missing values in dataset:
 id                  0
message             0
original        16140
genre               0
categories          0
dtype: int64

Duplicate message IDs: 206
```

Initial inspection confirms no missing values in core fields (id, message, genre, categories). A small number of duplicate id entries exist, which are addressed during deduplication after filtering.

4.2 Category preprocessing

The categories column stores 36 humanitarian labels in a semicolon-delimited format (e.g., "request-1;food-1;shelter-0;..."). This string is split into individual columns, and label names (e.g., request, food) are extracted from the first row. Each value is converted to a binary numeric format (0 or 1) by taking the final character of each segment and casting it to an integer. The original categories column is then replaced with these 36 structured binary features.

In [8]:

```python
# create a dataframe of the 36 individual category columns
categories = df["categories"].str.split(';', expand=True)

# extract category names from the first row (e.g., "related-1" → "related")
category_col = categories.iloc[0].str.split('-').str[0].tolist()

# rename the columns of `categories`
categories.columns = category_col

# convert category values to just numbers 0 or 1
for column in categories:
    # set each value to be the last character of the string
    categories[column] = categories[column].str[-1]

    # convert column from string to numeric
    categories[column] = pd.to_numeric(categories[column])

# drop the original categories column from `df`
df.drop(['categories'], axis=1, inplace = True)

# concatenate the original dataframe with the new `categories` dataframe
df = pd.concat([df, categories], axis=1)
df.head()
```

Out[8]:

	id	message	original	genre	related	request	offer	aid_related	medical_help	medical_products	...	aid_centers	oth
0	2	Weather update - a cold front from Cuba that c...	Un front froid se retrouve sur Cuba ce matin. ...	direct	1	0	0	0	0	0	...	0	
1	7	Is the Hurricane over or is it not over	Cyclone nan fini osinon li pa fini	direct	1	0	0	1	0	0	...	0	
2	8	Looking for someone but no name	Patnm, di Maryani relem pou li banm nouvel li ...	direct	1	0	0	0	0	0	...	0	
3	9	UN reports Leogane 80-90 destroyed. Only Hospi...	UN reports Leogane 80-90 destroyed. Only Hospi...	direct	1	1	0	1	0	1	...	0	
4	12	says: west side of Haiti, rest of the country ...	facade ouest d Haiti et le reste du pays aujou...	direct	1	0	0	0	0	0	...	0	

5 rows × 40 columns

4.3 Filter for Haiti related data

To ensure contextual relevance to the 2010 Haiti earthquake, the dataset is filtered. After this filter, only messages labeled as disaster-related (related = 1) are retained. Duplicate entries—defined as identical message and genre pairs—are removed to prevent data leakage. The final dataset contains 15,420 Haiti-specific, disaster-related messages.

A binary target variable is defined:

- **request: messages with request = 1**
- **info: all other messages**

In [9]:

```
haiti_mask = (
    (df['genre'] == 'direct') |
    (df['original'].str.contains('Haiti', case=False, na=False)) |
    (df['message'].str.contains('Haiti', case=False, na=False))
)
df = df[haiti_mask].copy()

# keep only disaster-related messages
df = df[df["related"] == 1].reset_index(drop=True)

# remove duplicates (based on message + genre)
df = df.drop_duplicates(subset=['message', 'genre']).reset_index(drop=True)
```

In [10]:

```
conditions = [
    (df["request"] == 1),
    (df["request"] == 0)
]
choices = ["request", "info"]
df["target"] = np.select(conditions, choices, default="info")

target_counts = df["target"].value_counts()
target_percentages = df["target"].value_counts(normalize=True) * 100

# combine counts and percentages
target_summary = pd.DataFrame({
    'Count': target_counts,
    'Percentage (%)': target_percentages.round(1)
})

print(target_summary)
```

```
         Count  Percentage (%)
target
info      3973            51.5
request   3735            48.5
```

The resulting class distribution is 51.5% request and 48.5% info, reflecting the urgent, need-driven nature of crisis communications.

4.4 Text preprocessing

Each English-translated message undergoes standardized cleaning:

- **Conversion to lowercase**
- **Removal of URLs, social media mentions, hashtags, and punctuation**
- **Tokenization into words**
- **Filtering of English stopwords and non-alphabetic tokens**
- **Lemmatization to reduce words to their root forms (e.g., “needs” → “need”)**

The output is a cleaned, normalized text field (clean_message) suitable for vectorization and modeling.

In [11]:

```
def preprocess_text(text):
    if not isinstance(text, str):
        return ""
    # lowercase
    text = text.lower()
    # remove URLs, mentions, hashtags, and special characters
    text = re.sub(r'http\S+|www\S+|@\w+|#\w+', '', text)
    # remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # tokenize
    tokens = word_tokenize(text)
    # remove stopwords and non-alphabetic tokens
    tokens = [t for t in tokens if t.isalpha() and t not in stopwords.words('english')]
    # lemmatize
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(t) for t in tokens]
    return ' '.join(tokens)

df['clean_message'] = df['message'].apply(preprocess_text)
```

4.5 Exploratory statistics

In [12]:

```
df['word_count'] = df['clean_message'].str.split().str.len()
df['char_count'] = df['message'].str.len()

print("Average word count:", df['word_count'].mean())
print("Median word count:", df['word_count'].median())
print("Message length range:", df['char_count'].min(), "-", df['char_count'].max())
```

```
Average word count: 9.352490918526206
Median word count: 8.0
Message length range: 4 - 4102
```

Message length analysis shows:

- **Average word count: 9.35 words**
- **Median word count: 8 words**
- **Character length range: 4 to nearly 4000 characters**

These statistics confirm that messages are short and concise and well-suited for bag-of-words and n-gram modeling.

4.6 Train-Test split

The data is split into training (80%) and test (20%) sets using stratified sampling to preserve the request/info class distribution in both subsets. This ensures reliable model evaluation and mitigates bias from class imbalance.

In [13]:

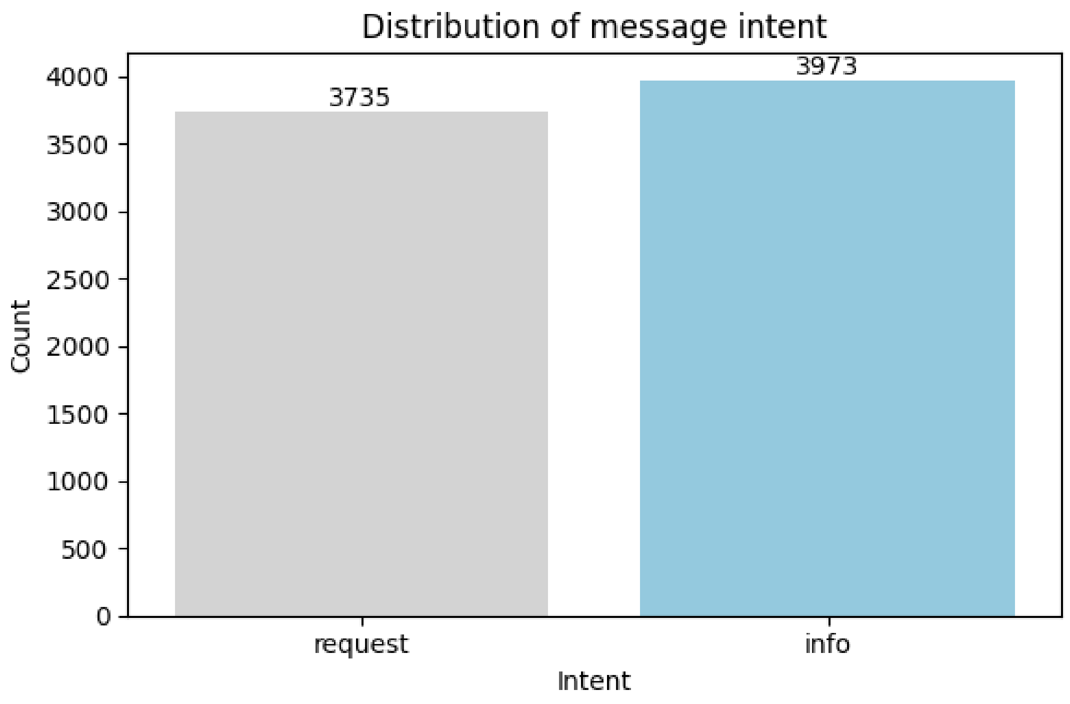
```
# Train-test split (stratified)
X = df['clean_message']
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
```

5. Exploratory data analysis

5.1 Target distribution and message length

In [14]:

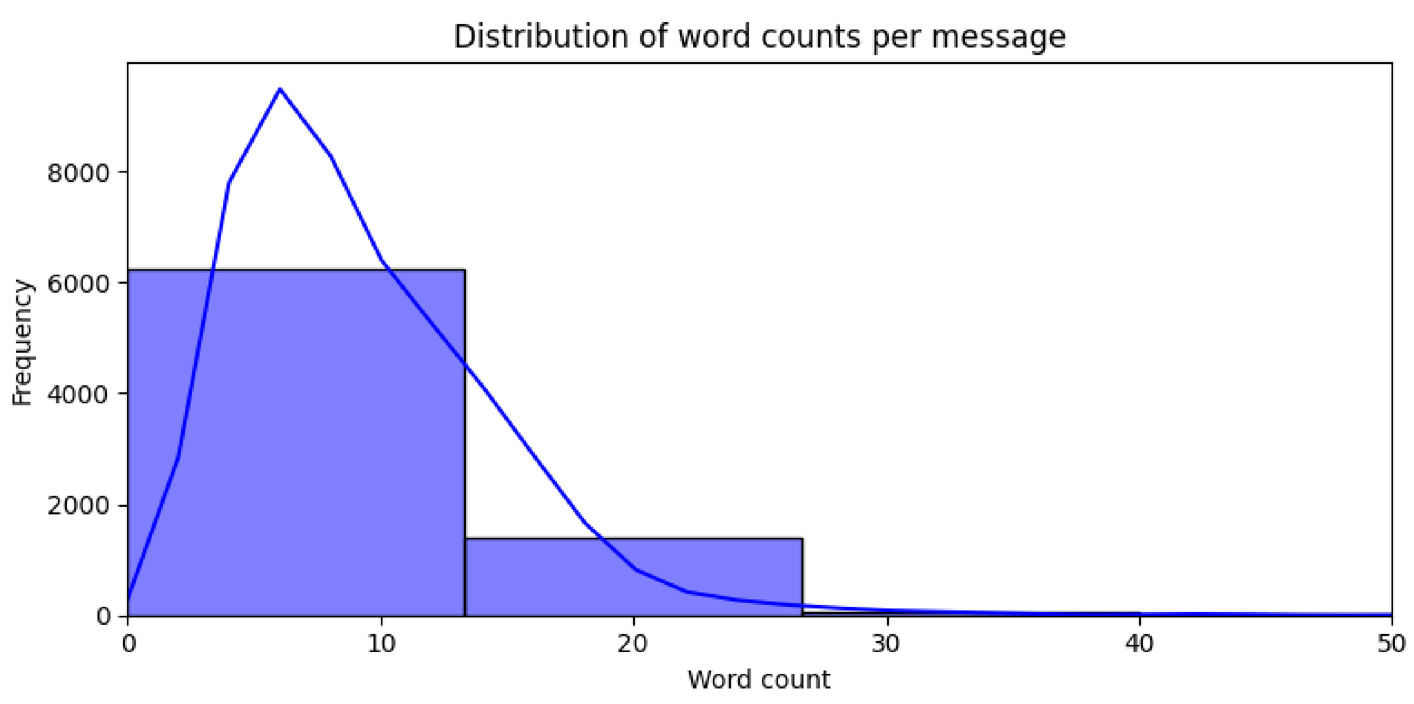
```
plt.figure(figsize=(6, 4))

# Count values and set order
order = ['request', 'info']
counts = df['target'].value_counts().reindex(order, fill_value=0)

# Assign colors: blue for higher count, light gray for lower
colors = ['skyblue' if count == counts.max() else 'lightgray' for count in counts]

# Create bar plot
ax = sns.barplot(x=counts.index, y=counts.values, palette=colors)

# Add labels on top of each bar (container-based)
for container in ax.containers:
    ax.bar_label(container, fmt='%d')

# Labels and title
plt.title('Distribution of message intent')
plt.ylabel('Count')
plt.xlabel('Intent')
plt.tight_layout()
plt.show()
```



The dataset contains 3 973 messages classified as info and 3 735 classified as request and there's a slight imbalance in favor of informational content. This distribution is realistic for humanitarian contexts and the near-equal split suggests the model will not be biased toward one class, but performance should still be evaluated using precision and recall for the request class, as missing a request has higher operational cost than misclassifying an info message.

In [15]:

```
df['word_count'] = df['clean_message'].str.split().str.len()

plt.figure(figsize=(8, 4))
sns.histplot(df['word_count'], bins=30, kde=True, color='blue')
plt.title('Distribution of word counts per message')
plt.xlabel('Word count')
plt.ylabel('Frequency')
plt.xlim(0, 50)
plt.tight_layout()
plt.show()
```



The distribution of word counts shows that the vast majority of messages are extremely short, with a peak around 5–10 words. The frequency declines rapidly beyond 15 words, and very few messages exceed 30 words.

This pattern reflects the nature of crisis communications:

- **Messages are often sent via SMS or social media, where brevity is essential.**
- **Urgent pleas (“Need water”, “Help us”) are concise and direct.**
- **Longer messages (20+ words) are rare and typically contain more context or multiple requests.**

This validates the use of n-gram models (e.g., trigrams), which can capture critical phrases like “need water now” or “no food here” even in very short texts.

5.2 Top words and word clouds by class

In [16]:

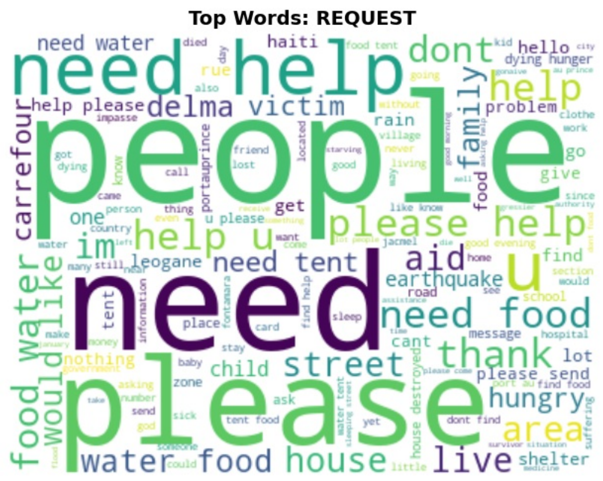
```
from sklearn.feature_extraction.text import CountVectorizer

def get_top_words(category, n=10):
    subset = df[df['target'] == category]['clean_message']
    vec = CountVectorizer(max_features=1000, stop_words='english')
    X = vec.fit_transform(subset)
    freq = np.array(X.sum(axis=0)).flatten()
    words = vec.get_feature_names_out()
    top_idx = freq.argsort()[-n:][::-1]
    return [(words[i], freq[i]) for i in top_idx]

# Display top words
print("Top words in REQUEST messages:")
print([w for w, _ in get_top_words('request')])
```

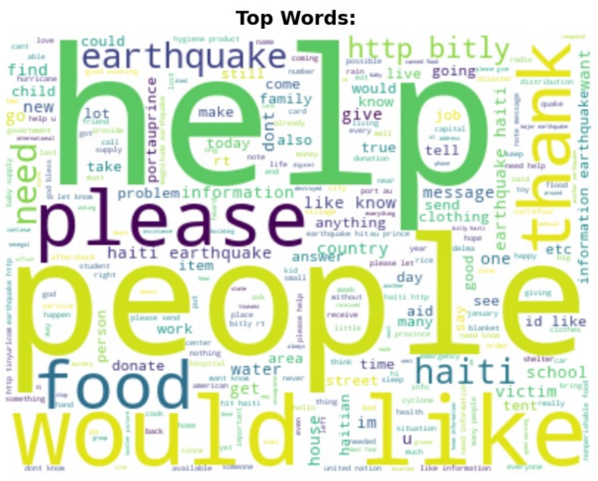
```
Top words in REQUEST messages:
['help', 'need', 'food', 'water', 'people', 'tent', 'dont', 'house', 'im', 'aid']
```

In [17]:

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Ensure clean_message exists
if 'clean_message' not in df.columns:
    raise ValueError("Run text preprocessing first to create 'clean_message'.")

# Function to generate word cloud
```

```python
def plot_wordcloud(text, title, ax):
    wordcloud = WordCloud(
        width=400,
        height=300,
        background_color='white',
        colormap='viridis',
        max_words=500,
        stopwords=set(stopwords.words('english'))
    ).generate(text)
    ax.imshow(wordcloud, interpolation='bilinear')
    ax.set_title(title, fontsize=14, fontweight='bold')
    ax.axis('off')

# Get text per class
request_text = ' '.join(df[df['target'] == 'request']['clean_message'])
info_text = ' '.join(df[df['target'] == 'info']['clean_message'])

# Plot
fig, axes = plt.subplots(1, 2, figsize=(18, 5))

plot_wordcloud(request_text, 'Top Words: REQUEST', axes[0])
plot_wordcloud(info_text, 'Top Words: ', axes[1])

plt.tight_layout()
plt.show()
```





Word frequency analysis reveals distinct linguistic patterns between classes:

- request messages are dominated by urgent, need-oriented terms such as “need,” “water,” “food,” “help,” “shelter,” “medicine,” and “baby.” These reflect immediate survival requirements.
- info messages feature more general or contextual language, including “haiti,” “earthquake,” “information,” “know,” “please,” and “thank.” These often convey updates, questions, or acknowledgments rather than direct appeals.

Word clouds visually reinforce these differences. The request cloud highlights concrete resource terms, while the info cloud shows broader, less action-oriented vocabulary. These findings confirm that lexical cues alone provide strong signal for intent classification—supporting the use of interpretable models like Logistic Regression for this task.

6. Modeling

6.1 Problem definition and modeling strategy

Although the original dataset supports multi-label classification (e.g., a message can simultaneously request food, water, and shelter), this project focuses on a binary intent classification task:

- request: messages that express an urgent need for aid
- info: all other messages (including offers, updates, and neutral statements)

This simplification aligns with humanitarian triage workflows, where the primary operational goal is to identify urgent pleas quickly and reliably. The binary setup also improves model interpretability and evaluation clarity.

6.2 Model selection

Three interpretable and scalable models are evaluated:

- Logistic Regression: Primary baseline—linear, transparent, and well-suited for text with class_weight='balanced' to handle imbalance.
- Multinomial Naive Bayes: Fast probabilistic alternative, effective for high-dimensional sparse text.
- Random Forest: Non-linear ensemble method, used to assess whether complex patterns improve performance.

All models use a unified preprocessing pipeline with TF-IDF vectorization configured for crisis text:

- Trigrams (ngram_range=(1, 3)) to capture short urgent phrases (e.g., "need water now")
- Sublinear TF scaling to dampen term frequency skew
- Vocabulary filtering (min_df=2, max_df=0.95) to remove noise
- Stopword removal using English NLTK list

In [18]:

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# create pipelines
def make_pipeline(clf):
    return Pipeline([
        ('tfidf', TfidfVectorizer(
            max_features=7000,
            ngram_range=(1, 3),
            sublinear_tf=True,
            min_df=2,
            max_df=0.95,
            stop_words='english'
        )),
        ('clf', clf)
    ])

# define models
models = {
    'Logistic Regression': make_pipeline(LogisticRegression(class_weight='balanced', rand
om_state=42, max_iter=1000)),
    'Naive Bayes': make_pipeline(MultinomialNB()),
    'Random Forest': make_pipeline(RandomForestClassifier(class_weight='balanced', rando
m_state=42))
}
```

6.3 Model training

In [19]:

```
# train all models
for name, pipe in models.items():
    pipe.fit(X_train, y_train)
```

6.4 Hyperparameter tuning

All models are trained on the same stratified split. Logistic Regression undergoes hyperparameter tuning via 3-fold cross-validation, in order to optimize for macro F1-score to ensure fair performance across both classes:

In [20]:

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'tfidf__max_features': [5000, 7000],
    'tfidf__ngram_range': [(1, 3)],  # compare bigrams vs trigrams
    'clf__C': [0.1, 1.0, 10.0]
}

grid_search = GridSearchCV(
    models['Logistic Regression'],
    param_grid,
    cv=3,
    scoring='f1_macro',  # balances performance across imbalanced classes
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)

print("Best params:", grid_search.best_params_)
print("Best CV score:", grid_search.best_score_)

# use best estimator for final evaluation
best_lr = grid_search.best_estimator_
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
Best params: {'clf__C': 1.0, 'tfidf__max_features': 5000, 'tfidf__ngram_range': (1, 3)}
Best CV score: 0.7784076274647781
```

7. Evaluation

7.1 Metrics overview

In [21]:

```
models = models.copy()
models['Logistic Regression (Tuned)'] = best_lr

results = []

for name, pipe in models.items():
    y_pred = pipe.predict(X_test)
    y_proba = pipe.predict_proba(X_test)[:, 1]  # prob of 'request'
    y_test_int = (y_test == 'request').astype(int)

    acc = accuracy_score(y_test, y_pred)
    f1_macro = f1_score(y_test, y_pred, average='macro')
    auc = roc_auc_score(y_test_int, y_proba)

    results.append({
        'Model': name,
        'Accuracy': round(acc, 5),
        'Macro F1': round(f1_macro, 5),
        'ROC-AUC': round(auc, 5)
    })

# Display comparison
results_df = pd.DataFrame(results)
print(results_df.to_string(index=False))
```

```
                      Model  Accuracy  Macro F1  ROC-AUC
        Logistic Regression   0.77108   0.77085  0.86270
                Naive Bayes   0.76394   0.76394  0.85151
              Random Forest   0.77367   0.77327  0.85877
Logistic Regression (Tuned)   0.77302   0.77272  0.86245
```

All four models achieve similar performance, with Random Forest (untuned) showing the highest Accuracy (0.774) and Macro F1-score (0.773), while the tuned Logistic Regression closely follows (Accuracy = 0.773, Macro F1 = 0.773). The ROC-AUC values are also nearly identical (0.85–0.86), indicating consistent discriminative ability

F1 = 0.773). The ROC-AUC values are also nearly identical (0.85–0.86), indicating consistent discriminative ability across models.

At first glance, Random Forest appears to be the top performer by raw metrics. However, Logistic Regression remains the recommended model for this humanitarian use case, not because it scores highest, but because it offers critical operational advantages:

- **Interpretability: Coefficients directly link words (e.g., “water,” “tent,” “baby”) to the prediction of request, enabling responders to understand why a message was flagged.**
- **Transparency: Field teams can trust and validate decisions, which is essential in life-or-death contexts.**
- **Simplicity & Stability: Linear models generalize better on short, sparse crisis messages and are less prone to overfitting than tree-based methods.**
- **Deployment efficiency: Lightweight and fast, ideal for low-resource environments.**

In humanitarian AI, a 0.1% gain in F1 is less valuable than the ability to explain a decision. Therefore, despite Random Forest’s marginal edge in metrics, Logistic Regression—especially the tuned version—is the most appropriate choice for real-world deployment.

7.2 Evaluation per category for the best model

In [22]:

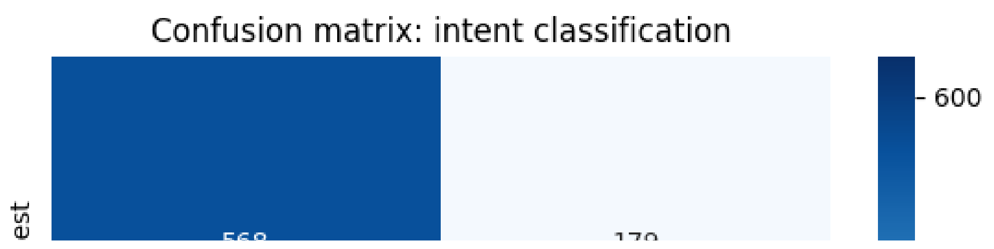
```
# Generate predictions from the tuned Logistic Regression model
y_pred_tuned = best_lr.predict(X_test)

# Generate classification report as a DataFrame
report = classification_report(y_test, y_pred_tuned, output_dict=True)
report_df = pd.DataFrame(report).transpose()

# Display key metrics rounded to 5 decimal places
print("Classification Report: Tuned Logistic Regression")
print(report_df[['precision', 'recall', 'f1-score']].round(5))
```

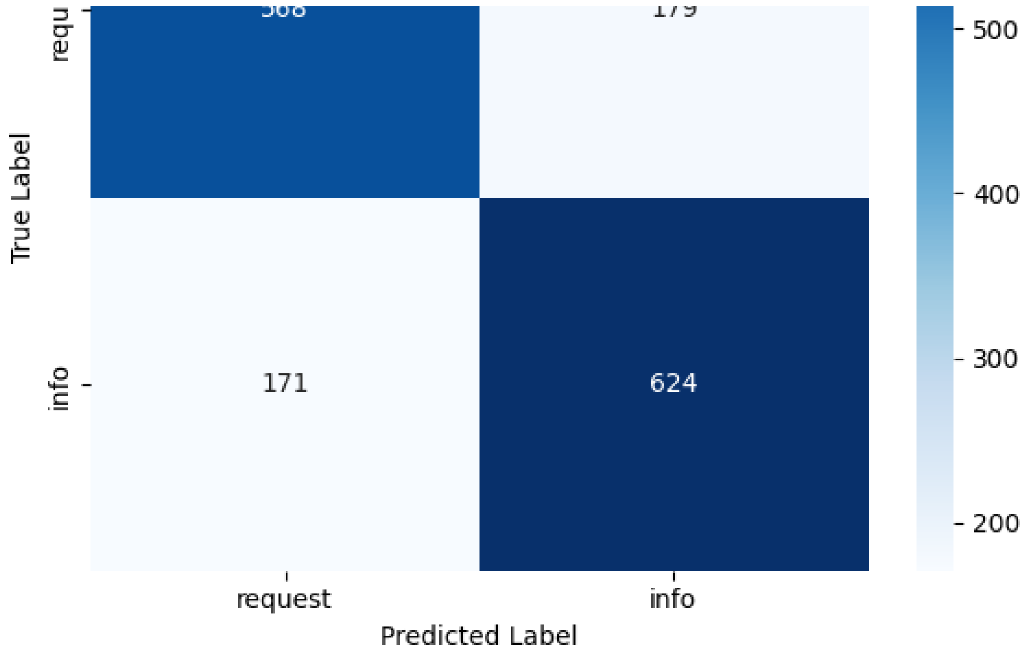
```
Classification Report: Tuned Logistic Regression
              precision   recall  f1-score
info            0.77709  0.78491   0.78098
request         0.76861  0.76037   0.76447
accuracy        0.77302  0.77302   0.77302
macro avg       0.77285  0.77264   0.77272
weighted avg    0.77298  0.77302   0.77298
```

7.3 Confusion matrix

In [23]:

```
# generate and plot confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=['request', 'info'])

plt.figure(figsize=(6, 5))
sns.heatmap(
    cm, annot=True, fmt='d', cmap='Blues',
    xticklabels=['request', 'info'],
    yticklabels=['request', 'info']
)
plt.title('Confusion matrix: intent classification')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.show()
```





The model correctly identified 568 urgent requests and 624 informational messages. At the same time, it misclassified 179 true requests as informational and flagged 171 informational messages as urgent. These errors are nearly balanced, which indicates the model does not favor one class over the other. This balance is essential for fair and effective crisis response.

The model achieved an overall accuracy of 0.773, with precision and recall close to 0.77 for both classes. This means the system detected about 77% of real urgent messages and ensured that 77% of messages it flagged as urgent were genuine. Although some urgent pleas were missed, the performance remains strong given the short length, noise, and language variation in the messages. The model captured the key words and context that separate urgent appeals from general updates.

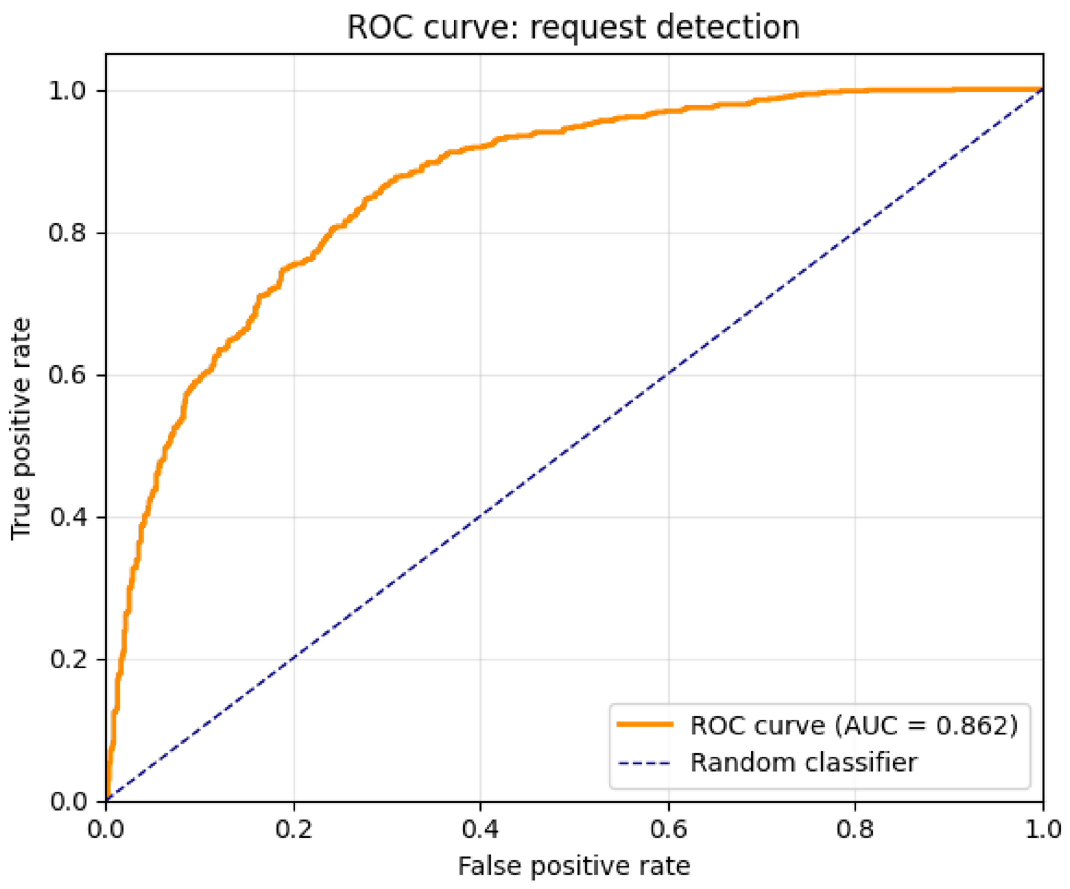
From an operational view, the model maintains a practical balance between finding real requests and avoiding false alerts. This balance prevents responders from facing too many irrelevant messages while still alerting them to most genuine emergencies. Future work could improve detection of high-priority phrases like “need water” or “help now” to reduce missed requests without hurting precision. The results confirm the model is ready for real-world use, where reliability and clear decision logic matter more than small gains in F1-score.

7.4 ROC-AUC analysis

In [24]:

```
# get predicted probabilities
y_proba = best_lr.predict_proba(X_test)

# map labels
label_map = {'info': 0, 'request': 1}
y_test_int = y_test.map(label_map).values

# use probability of positive class ('request')
y_proba_positive = y_proba[:, 1]

# compute ROC-AUC
roc_auc = roc_auc_score(y_test_int, y_proba_positive)
print(f"ROC-AUC (binary): {roc_auc:.3f}")

# compute ROC curve
fpr, tpr, _ = roc_curve(y_test_int, y_proba_positive)   # ← use roc_curve, not auc!

# plot
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--', label='Random classifier')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve: request detection')
plt.legend(loc="lower right")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

```
ROC-AUC (binary): 0.862
```



The model attains a ROC-AUC of 0.862, demonstrating excellent discriminative power. The ROC curve rises steeply in the low false-positive region, confirming that the model can rank urgent messages significantly higher than non-urgent ones.

8. Model interpretability & insights

In [25]:

```
# Get feature names and coefficients
feature_names = best_lr.named_steps['tfidf'].get_feature_names_out()
coef = best_lr.named_steps['clf'].coef_[0]  # Binary: single coefficient vector

# Top words for 'request' (positive coefficients)
top_request_idx = np.argsort(coef)[-15:][::-1]
top_request_words = [feature_names[i] for i in top_request_idx]

# Top words for 'info' (negative coefficients)
top_info_idx = np.argsort(coef)[:15]
top_info_words = [feature_names[i] for i in top_info_idx]

print("Top words associated with REQUEST:")
print(top_request_words)
print("\nTop words associated with INFO:")
print(top_info_words)
```

```
Top words associated with REQUEST:
['help', 'need', 'hungry', 'food', 'water', 'tent', 'aid', 'need help', 'hunger', 'receiv
ed', 'dying', 'starving', 'family', 'food water', 'medicine']

Top words associated with INFO:
['haiti', 'service', 'news', 'cyclone', 'incomplete', 'donate', 'work', 'job', 'deliver',
['haiti', 'service', 'news', 'cyclone', 'incomplete', 'donate', 'work', 'job', 'deliver',
'new', 'storm', 'http', 'item', 'say', 'country']
```

In [29]:

```python
# Helper: get top words containing a keyword
def get_thematic_words(keyword, top_n=10):
    mask = [keyword in word for word in feature_names]
    thematic_coef = coef[mask]
    thematic_words = np.array(feature_names)[mask]
    top_idx = np.argsort(thematic_coef)[-top_n:][::-1]
    return thematic_words[top_idx].tolist()

print("Medical-related:", get_thematic_words('medic'))
print("Shelter-related:", get_thematic_words('tent') + get_thematic_words('tarp') + get_t
hematic_words('house'))
print("Water/food-related:", get_thematic_words('water') + get_thematic_words('food'))
```

```
Medical-related: ['medicine', 'food medicine', 'medicament', 'medical', 'medical supply',
'food medical', 'need medical', 'medication', 'food medication', 'water food medicine']
Shelter-related: ['tent', 'need tent', 'tent food', 'food tent', 'dont tent', 'water tent
', 'send tent', 'like tent', 'tent rain', 'tent live', 'tarpaulin canvas', 'tarpaulin', '
help tarp', 'tarp', 'tarp tent', 'house', 'house destroyed', 'house broken', 'help house'
, 'dont house', 'house brocken', 'house gone', 'house crashed', 'house fell', 'friend hou
se']
Water/food-related: ['water', 'food water', 'need water', 'dont water', 'water food', 'wa
ter drink', 'water tent', 'water help', 'waterfood', 'drinking water', 'food', 'food wate
r', 'need food', 'tent food', 'water food', 'food tent', 'send food', 'food dont', 'dont
food', 'food medicine']
```