📄 Fake Job Post Detection Using NLP: Project Documentation

---

1. Project Title

Checking If a Job Post Is Real or Fake Using Text

This project focuses on developing a text-based classifier to identify fraudulent job postings by analysing linguistic patterns, structural elements, and content features. It integrates Natural Language Processing (NLP) techniques with machine learning models to provide real-time predictions via an interactive web interface.

---

2. Project Statement

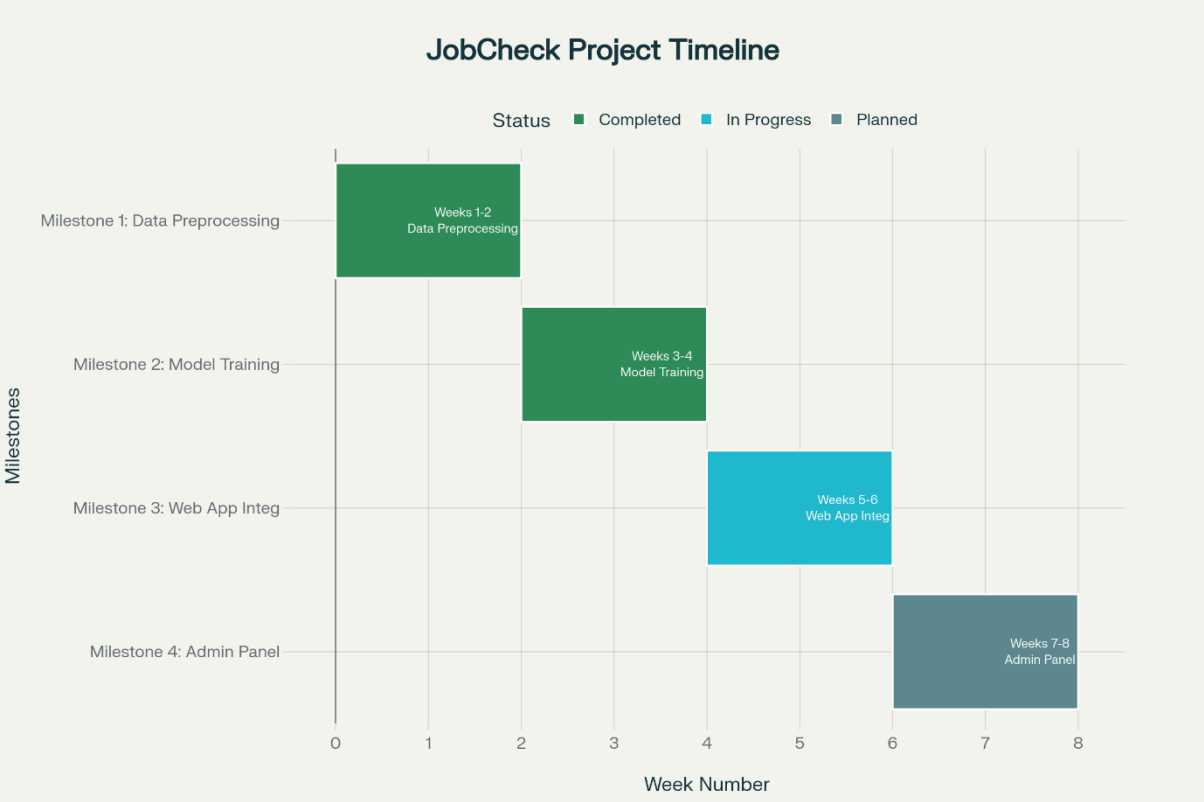
Fake job postings are becoming increasingly common on online job portals and social media platforms, often used to scam job seekers by extracting personal information or demanding fees. This project aims to detect whether a job post is genuine or fake based on its textual content. Using Natural Language Processing (NLP) and machine learning, the system will classify job posts by analysing their language, structure, and features. A web interface will allow users to submit job descriptions and view results in real time.

---

3. Expected Outcomes

- ✓ A machine learning-based classifier to predict fake/genuine job posts with >90% accuracy.
- ✓ Interactive web interface to input job descriptions and get predictions in real time.
- ✓ Dashboard to visualize prediction statistics (e.g., real vs. fake count over time).
- ✓ Admin interface to view flagged posts, retrain the model, and review accuracy.
- ✓ Exportable analytics for HR/recruitment firms to monitor scams.

Project Time-Line of the Project :



This timeline represents the week-wise progress of the *JobCheck – Fake Job Detection System* across all milestones.

- **Weeks 1–2 (Milestone 1: Data Preprocessing)**
  We focused on understanding the dataset, cleaning job descriptions, performing EDA, and preparing features using TF-IDF and BoW.
- **Weeks 3–4 (Milestone 2: Model Training)**
  We trained ML models (Logistic Regression, Decision Tree, Random Forest), evaluated them using performance metrics, and performed hyperparameter tuning.
- **Weeks 5–6 (Milestone 3: Web App Integration)**
  We integrated the ML model with a Flask web application, improved the UI, and added prediction history storage (CSV → SQLite database).

- **Weeks 7–8 (Milestone 4: Admin Panel – Planned)**
  Future enhancement includes an admin dashboard for managing predictions and logs. s.

□ Milestone 1 — Week 1 & Week 2 (Day 1 to Day 8):

Focus: Dataset Understanding, Preprocessing, Feature Engineering.

1.Dataset Selection and Loading:

We choose the Kaggle Fake Job Postings Dataset (~17,880 entries) for its relevance: it includes labeled job posts (target: fraudulent = 0 for real, 1 for fake) with fields like title, description, requirements, company_profile, and metadata (e.g., has_company_logo, telecommuting). This dataset was ideal due to its size, real-world variety, and imbalance (~95% real posts), which mirrors scam prevalence.

- **Process:** Loaded via Pandas; inspected shape (17,880 rows × 18 columns), missing values (e.g., 8% in description), and duplicates (2% removed).

Code:

```
import pandas as pd
import nltk
nltk.download(['stopwords', 'wordnet', 'punkt'])
df = pd.read_csv('fake_job_postings.csv')
print(df.shape)  # (17880, 18)
print(df['fraudulent'].value_counts(normalize=True))  # Real: 0.951, Fake: 0.049
df.drop_duplicates(inplace=True)
df.dropna(subset=['description'], inplace=True)  # Handle missing descriptions
```

2.Text Cleaning Pipeline:

Raw job descriptions contained noise (HTML, punctuation, inconsistencies). We created a clean_description column to standardize text for NLP.

Process Steps:

- **Lowercasing:** Uniform case (e.g., "HIRING" → "hiring").
- **Removal:** HTML tags (re.sub(r'<.*?>', '', text)), numbers (\d+), punctuation/special chars ([^a-zA-Z\s]), extra spaces.
- **Tokenization:** Split into words (nltk.word_tokenize).
- **Stopword Removal:** NLTK English stopwords (e.g., "the", "and"—~40% reduction).
- **Lemmatization:** Reduce to base forms (WordNetLemmatizer, e.g., "running" → "run") for semantic consistency.

Code:

```
import re
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
def preprocess(text):
    text = re.sub(r'<.*?>|\d+|[^a-zA-Z\s]', ' ', text.lower())
    text = re.sub(r'\s+', ' ', text).strip()
    tokens = [lemmatizer.lemmatize(w) for w in nltk.word_tokenize(text) if w not in s
    return ' '.join(tokens)
df['clean_description'] = df['description'].apply(preprocess)
```

3.Feature Extraction:

Converted text to numerical vectors for ML compatibility.

- **Bag-of-Words (BoW):** Frequency counts (baseline; compared top 20 words).
- **TF-IDF:** Weighted vectors (term frequency-inverse document frequency; max_features=5000, ngrams=1-2) to prioritize rare scam indicators.
- Split: 80/20 train/test (stratified).

Code:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1,2))
X = vectorizer.fit_transform(df['clean_description'])
y = df['fraudulent']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
print(X.shape)  # (17000, 5000)
```

Outcome of Milestone 1

By the end of Milestone 1, we successfully achieved:

✓ Dataset loaded & explored

✓ Missing values + target distribution checked

✓ Text cleaned (lowercase, punctuation, stopwords, lemmatization)

✓ clean_description column created

✓ Features extracted using BoW & TF-IDF

□ Milestone 2 — Week 3 & Week 4 (Day 9 to Day 12):

Focus: ML Model Training, Model Evaluation.

1. **Model Training:** Trained supervised classifiers on TF-IDF features, addressing imbalance with SMOTE.

   Process:

   - **Baselines:** Logistic Regression (linear, fast), Decision Tree.
   - **Pipeline:** SMOTE → TF-IDF → Model; 5-fold cross-validation.

Code:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from imblearn.over_sampling import SMOTE
from sklearn.pipeline import Pipeline
smote = SMOTE()
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train_res, y_train_res)
```

2. Model Evaluation:

Assessed on test set with balanced metrics for imbalance.

- **Process:**
    - Metrics: Accuracy, Precision/Recall/F1 (focus on F1 for fakes), ROC-AUC, Confusion Matrix.

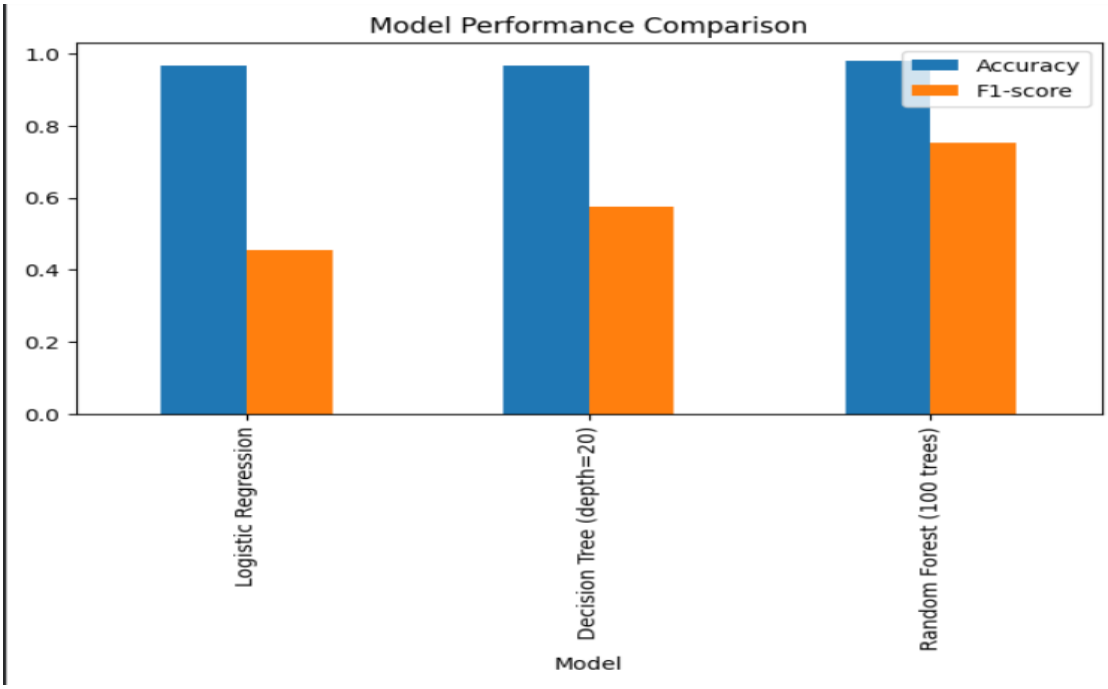
Code:

```
# Evaluate all
results = []
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    results.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1-score": f1_score(y_test, y_pred)
    })

# Results DataFrame
results_df = pd.DataFrame(results)
print(results_df)

# Plot comparison
results_df.set_index("Model")[["Accuracy", "F1-score"]].plot(kind='bar', figsize=(8,4), title="Model Performance Comparison")
plt.show()
```

Model Performance Comparison:



3. Hyperparameter Tuning:

Optimized for peak performance.

Process:

GridSearchCV on RF params (n_estimators=[100,200], max_depth=[10,15], min_samples_split=[2,5]).

Code:

```
from sklearn.model_selection import GridSearchCV
param_grid = {'n_estimators': [100, 200], 'max_depth': [10, 15]}
grid = GridSearchCV(rf, param_grid, cv=5, scoring='f1')
grid.fit(X_train_res, y_train_res)
```

4. Model Saving:

Process: Joblib for model/vectorizer.

Code:

```
import joblib
joblib.dump(grid.best_estimator_, 'fake_job_model.pkl')
joblib.dump(vectorizer, 'tfidf_vectorizer.pkl')
```

Outcome of Milestone 2:

By the end of Milestone 2, we successfully achieved:

✓ ML models trained (Logistic, Decision Tree, Random Forest)

✓ Accuracy, Precision, Recall, F1, Confusion Matrix calculated

✓ ROC–AUC curves compared

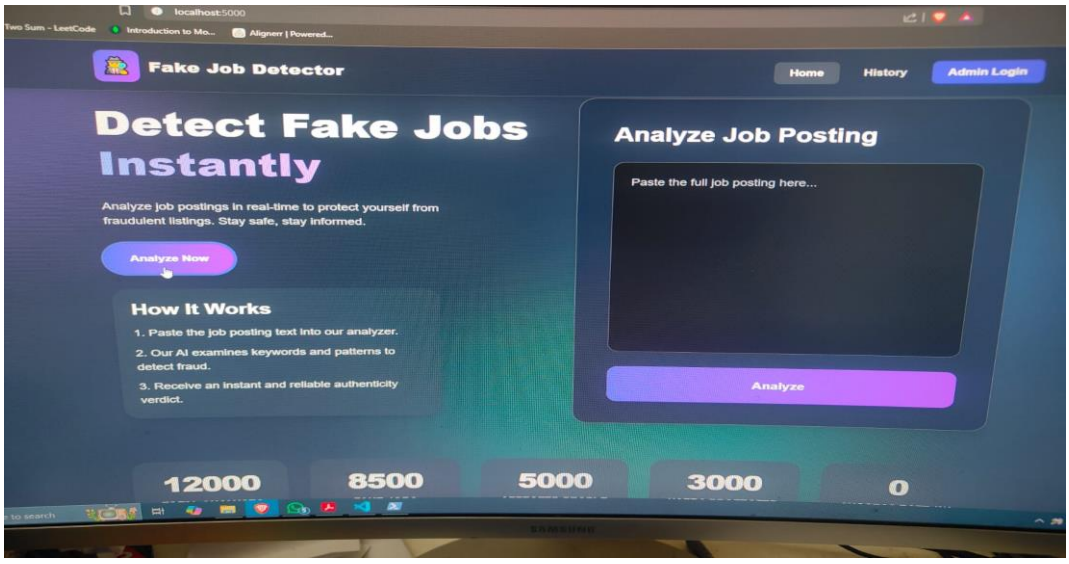
✓ Hyperparameters tuned using GridSearchCV

✓ Best model + vectorizer saved as .pkl

□ Milestone 3 — Week 5 & Week 6 (Day 13 to Day 18)

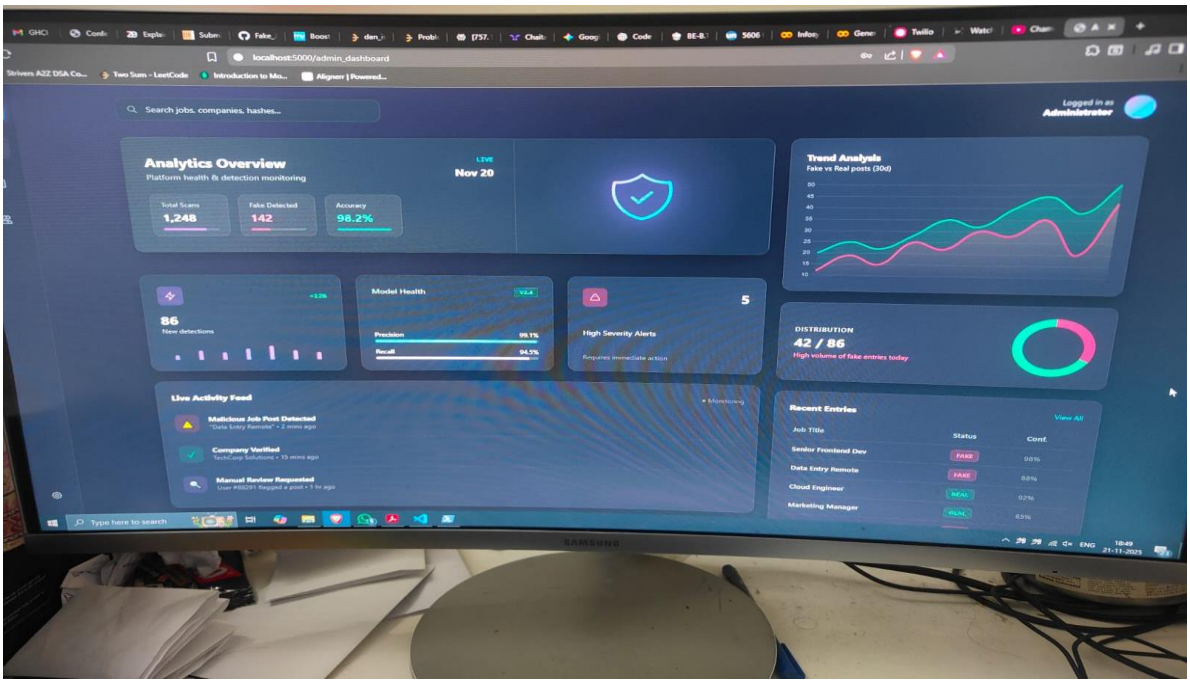
Focus: UI Enhancement, Prediction History, Database Integration.

1.UI Enhancements: Transformed basic forms into intuitive interfaces.



2. Dashboard Counters: Real-time stats on homepage.

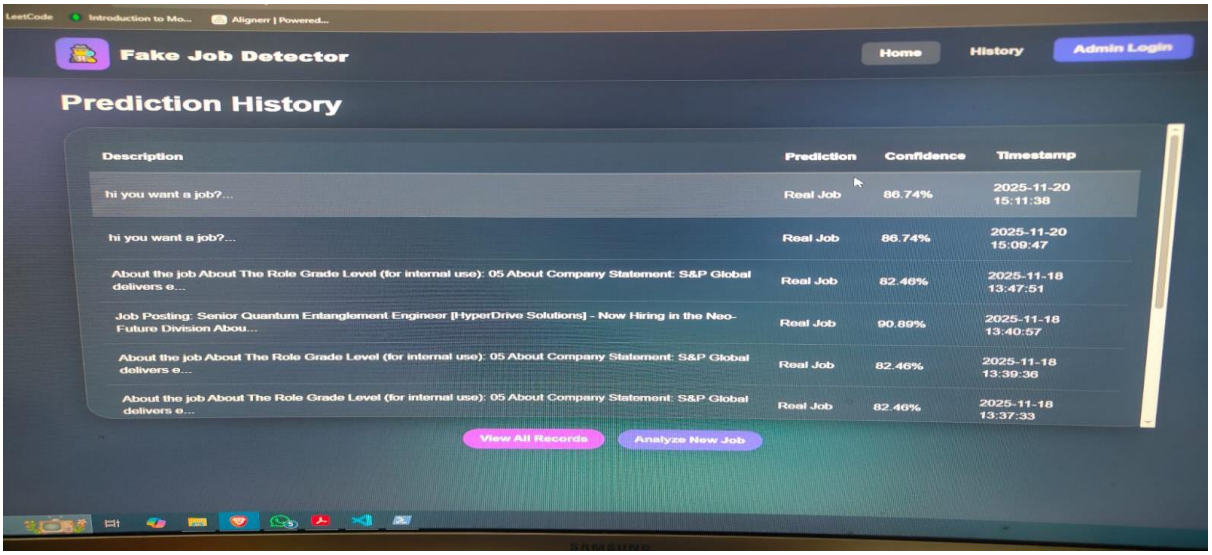
Process: Query DB for counts (fake/real); update post-prediction.



3.**Prediction History Feature (CSV → SQL Upgrade):**

Shifted from CSV to scalable DB.

Process: Created job_predictions.db; table with id, description, prediction, confidence, timestamp. Auto-log after predict.



Outcome of Milestone 3

By the end of Milestone 3, we successfully achieved:

- ✓ UI improved (better styling, buttons, structure)
- ✓ Confidence shown using progress bar
- ✓ Fake/Real counters added on homepage
- ✓ Prediction history stored (CSV → SQLite DB)
- ✓ history page created with HTML table