CS657 October 24, 2024

HW2:Predicting Fake Job Posting

Antara Tewary (G01413546),Ankit Kumar (G01436204)

1. **Introduction**

   This assignment aims to create a system that flags suspicious job postings on Indeed.com, addressing the challenge of fraudulent postings that mislead the efforts of job seekers. We are using Fake Job Prediction Dataset, which contains 18000 job descriptions out of which 800 are fraudulent samples. The goal is to create an automated model which prioritizes which posting to review based on the likelihood of it being fraudulent.

2. **Data Preparation**

   - *Label Encoding and Filtering Invalid Data*: The target column fraudulent was cast to double type to represent legitimate (0) and fake (1) job postings. Any records with invalid values were filtered out.
   - *Handling Missing Values*:We identified missing values by calculating percentage of null or NaN values across columns and dropped those with more than 1% null values to reduce noise.
   - *Text Cleaning*: The job descriptions were cleaned by removing special characters, converting text to lowercase, and removing stop words to ensure uniformity.
   - *Balancing the Dataset*: To address class imbalance, we implemented a custom balancing strategy:
     - First, we separated and cached the fraudulent postings (minority class)
     - Calculated the sampling fraction by dividing the count of fraudulent postings by non-fraudulent postings
     - Used this fraction to randomly undersample the majority class (legitimate postings) without replacement
     - Combined the minority class with the sampled majority class using union operation
     - Randomly shuffled the final balanced dataset to ensure random distribution

     This approach ensured a perfectly balanced dataset while maintaining all fraudulent cases, enabling unbiased model training. The caching of the minority class and random shuffling helped optimize performance and prevent any ordering bias in the final dataset.

Here are some statistics of the dataset after cleaning:

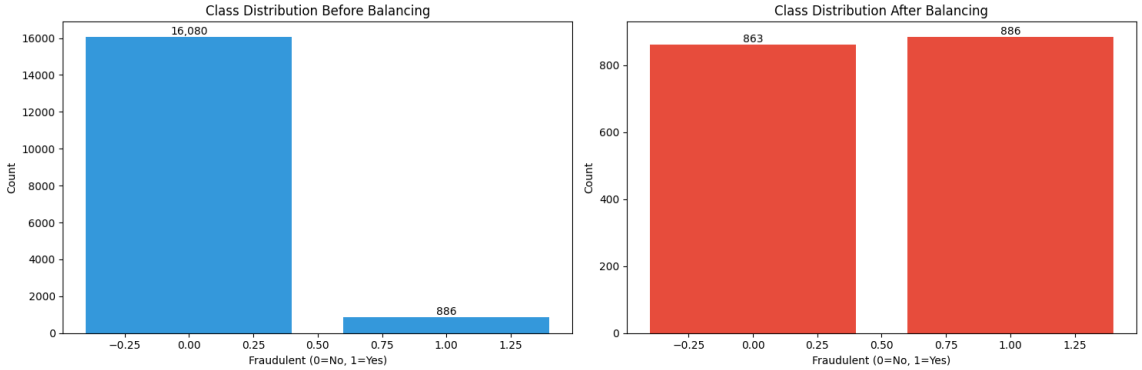


Figure 1: Class distribution before and after balancing

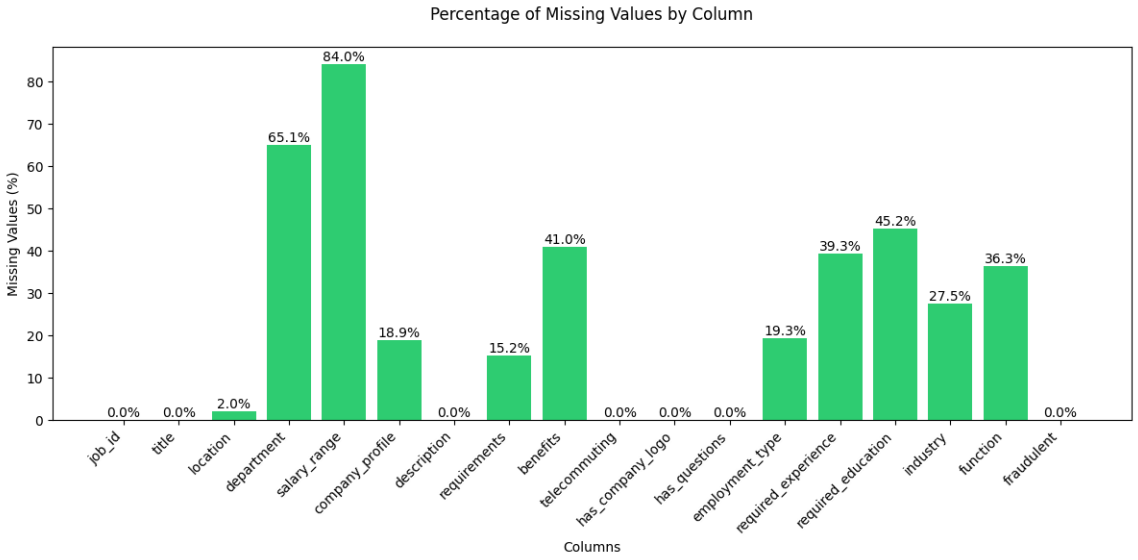


Figure 2: % of missing values for each column

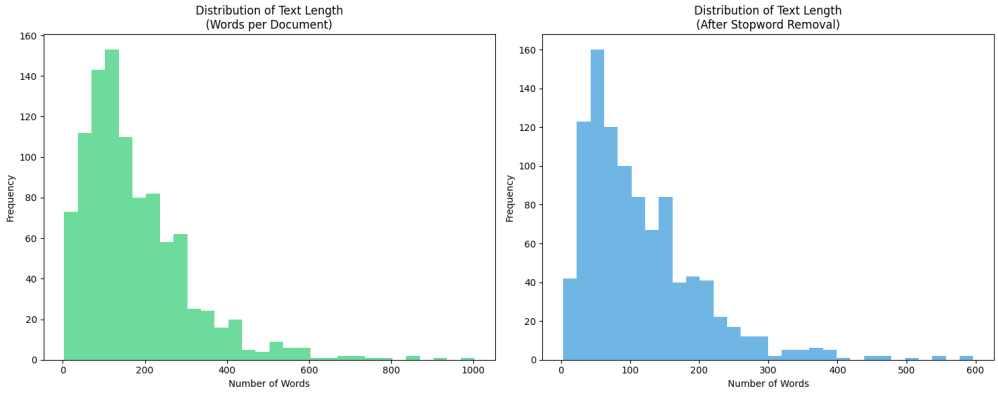
3. **Feature Engineering**

The data contains text features that need to be transformed into numerical representations for model training. We implemented a custom pipeline for processing the job posting text data:

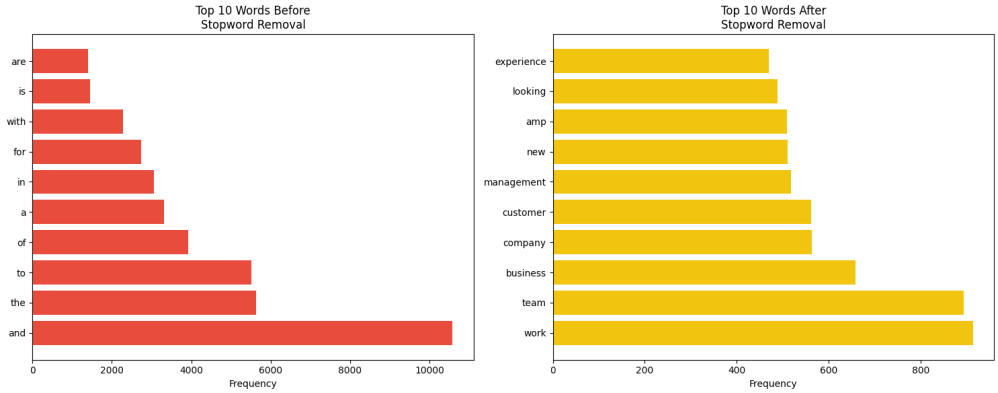
- **Separate Text Processing**: We processed two key text fields independently,recognizing their different contextual importance.
  Description Processing:
  - Tokenization splits descriptions into individual words, enabling word-level analysis
  - Stopword removal eliminates common words (e.g., "the", "and"), reducing noise
  - Count vectorization with a 5000-term vocabulary captures the rich detail in descriptions
  - Minimum document frequency of 2.0 eliminates rare terms that might represent noise
    Impact: This approach preserved the detailed information in job descriptions while reducing dimensionality from unlimited text to a focused 5000-dimension vector.

  Title Processing:
  - Applied same tokenization and stopword removal
  - Used a compact 15-term vocabulary using count vectorizer, focusing on key job title terms
  - Maintained minimum document frequency of 2.0 for consistency
    Impact: This concentrated representation captures essential job title information without overwhelming the more detailed description features.
- **Feature Combination and Scaling**
  - VectorAssembler combined title (15 dimensions) and description (5000 dimensions) vectors
  - StandardScaler normalized feature magnitudes while preserving zero elements
  - Disabled mean centering to maintain vector sparsity
    Impact: This approach:
    * Maintains the relative importance of titles vs descriptions
    * Ensures all features contribute proportionally to model decisions
    * Preserves memory efficiency through sparse vector representation

Some statistics after text transformation are shown below:



(a) Text length comparison before and after stopword removal



(b) Top words before and after stopword removal

Figure 3: Text Transformation Statistics

4. **Model Training and Tuning**

We split the dataset was split into training (70%) and test (30%) sets using PySpark's randomSplit function with a seed value of 42 for reproducibility. The dataFrames were cached to optimize performance during model training.

- *Logistic Regression*
  - Trained using a binary classification approach.
  - Key challenges included handling high-dimensional text features.
  - Used regularization parameters to prevent overfitting.
  - Parameters tuned: regularization parameter (0.01-0.3), elastic net mixing (0.0-1.0), and maximum iterations (10-100)
- *Linear SVC*
  - Implemented for binary classification of job postings
  - Focused on finding the optimal hyperplane for classification
  - Parameters tuned: regularization (0.1-1.0), maximum iterations (10-100), and tolerance (1e-3, 1e-4)
- *Random Forest Classifier*
  - Ensemble learning method using multiple decision trees.
  - Handled high-dimensional feature space effectively
  - Parameters tuned: number of trees (10-30), maximum depth (5-15), and impurity criterion (gini, entropy)
- *Multilayer Perceptron Classifier*
  - Neural network with architecture [num_features, 20, 2]
  - Designed for complex pattern recognition in text data Parameters tuned: maximum iterations (50, 100), block size (64, 128), and tolerance (1e-4, 1e-5)
- *Hyperparameter Tuning*
  - Implemented 10-fold cross-validation for all models

- Used ParamGridBuilder for systematic parameter search
- Selected hyperparameters focused on:
  - Controlling model complexity (regularization parameters)
  - Optimization settings (iterations, tolerance)
  - Model-specific parameters (trees, depth for Random Forest)
- Evaluation metrics: F1 score used as primary metric for cross-validation
- Each model was evaluated on both training and test sets to ensure generalization

5. **Model Training and Cross-validation**
We implemented a comprehensive evaluation of four machine learning models using 10-fold cross-validation on the training set (70% of data) and final testing on the held-out test set (30% of data). Each model was tuned using grid search for optimal hyperparameters.

- *Model Performance Summary*:

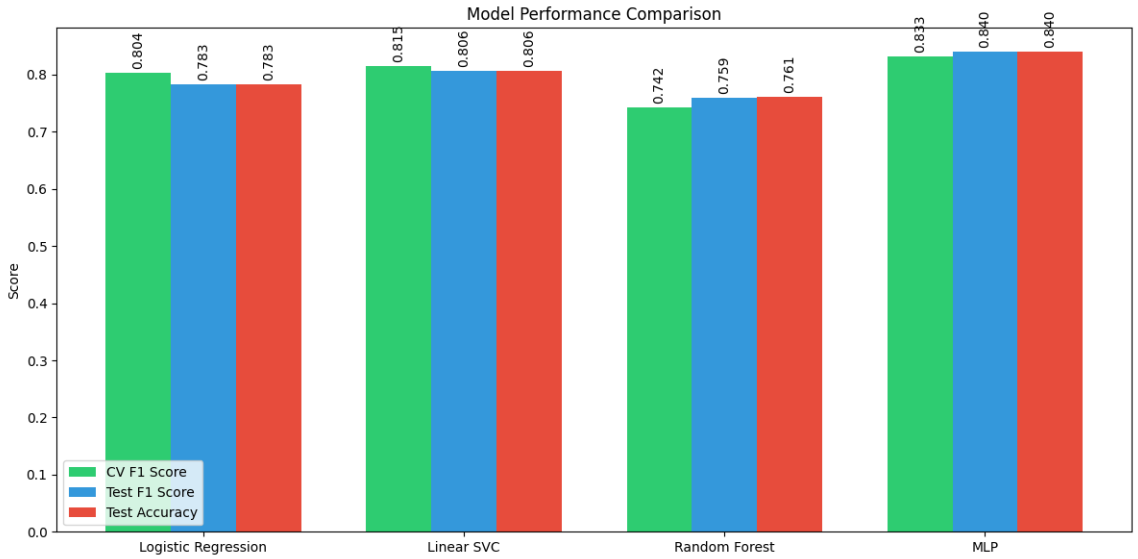


Figure 4: Performance Comparison of Different Models

Model	CV F1	Test F1	Accuracy	Best Parameters
Logistic Regression	0.804	0.783	0.783	regParam=0.0, elasticNet=0.0, maxIter=100
Linear SVC	0.815	0.806	0.806	regParam=0.0, maxIter=100, tol=1e-6
Random Forest	0.742	0.760	0.761	numTrees=20, maxDepth=5, impurity=gini
MLP Classifier	**0.833**	**0.840**	**0.840**	maxIter=100, blockSize=128, tol=1e-6

Table 1: Model Performance and Optimal Parameters

6. **Conclusion**

- **Best Performing Model**: The Multilayer Perceptron Classifier achieved the highest performance with test F1 score and accuracy of 0.840.
- **Model Comparison**:
  - MLP and Linear SVC showed superior performance ($F1 > 0.80$)
  - Logistic Regression performed moderately well (F1 $\approx$ 0.78)
  - Random Forest showed slightly lower performance (F1 $\approx$ 0.76)
- **Consistency**: All models showed good consistency between cross-validation and test performance, indicating robust generalization.
- **Key Insights**
  - Neural network-based approach (MLP) demonstrated superior capability in capturing complex patterns in job posting text

- Tree-based ensemble method (Random Forest) showed robust performance with simpler implementation
- Linear models (SVC and Logistic Regression) performed adequately but were limited in capturing complex text patterns
- All models achieved relatively consistent accuracy and F1 score, indicating balanced prediction capabilities