Fake Job Recruitment Detection Using Classification

https://drive.google.com/file/d/1fjlc6TQjCRrO7e
b9AhUE2WrnQeCPupwm/view?usp=share_link

Ethan Rangel-Torres

BRIEF:

The goal of this project is to develop a POC detection model to distinguish if a job recruitment is real or fake. This will be achieved testing 3 classification models (Logistic Regression, LightGBM, XGBoost).

01

Problem & Challenges

02

Motivation & Different Approaches

03

Methods & Experiments

04

Results & Observations

05

Conclusion & Future Work

01

Problems & Challenges

Project Background

Fraudulent Job Postings:

- ➔ This project will be a binary classification on whether a job posting is real or fake.
- ➔ When making classification models on fraud, many use False Negatives as the main deciding factor because it is better to be safe when dealing with conclusions like detecting cancer or financial information.
- ➔ We can see fraudulent classification in our everyday lives when certain emails get sent to junk based on whether they seem like a scam or real.
- ➔ However, some of these emails can still make it through this detection system and could even be fake job opportunities looking to get personal information.

Challenges

Challenges detecting fraud:

- ➔ High quality data is crucial for training reliable fraud detection models. Balancing the training set is especially important because without it, identifying scams becomes significantly more difficult.
- ➔ Misclassifying a scam as legitimate can be dangerous, especially when applicants submit sensitive personal information.
- ➔ At the same time, over-predicting fraud can lead to genuine opportunities being unfairly filtered out, so striking the right balance is essential.

02

Motivation & Different Approaches

Motivation

Fake job postings waste time in this difficult job market.

The main motivation is to create a POC that can be further implemented by bigger development teams to create a tool for applicants to use.

I have personally dealt with fake recruitments on LinkedIn or Indeed and know its a common problem employment seekers face. The job market is currently difficult for new graduates and weeding out fake recruitments will help applicants spend more time on real opportunities.

Existing Approaches

Traditional Methods:

- ➔ Logistic Regression, Decision Trees, Random Forest, and Naive Bayes are some of the most commonly used classification models in fraud detection.
- ➔ These models are popular due to their simplicity, interpretability, and low computational cost.
- ➔ They are extremely beneficial because they can provide a good starting point or baseline before testing more complex method.

Existing Approaches

Limitations:

- ➔ Many struggle with imbalanced datasets, often leading to high false negatives in fraud detection.
- ➔ Models like Logistic Regression assume linear relationships, which may not hold in complex fraud scenarios.
- ➔ These models may lack the ability to capture hidden patterns or nonlinear relationships found in real-world data. Often require manual feature engineering to be effective.

03

Methods & Experiments

1. Unbalanced and Raw Data
2. Cleaning, Preprocessing, & Feature selection
3. Train-Test Split + TF-IDF + SMOTE
4. Train Baseline Models
5. Evaluate and Select Best 2 Models
6. Use GridSearch to Parameter tune
7. Evaluate best model

Why use this different approach?

Why use multiple Classification models with hyperparameter tuning?

- ➔ The paper, “Fake Job Recruitment Detection Using Machine Learning Approach” used traditional classifiers such as Naive Bayes, Support Vector Machines, Decision Trees, and Random Forests to evaluate performance on the same dataset.
- ➔ I took a similar but unique approach by experimenting with models known for high performance in real-world text classification tasks (Logistic Regression, LightGBM, and XGBoost). Using hyperparameter tuning and data class balancing techniques allowed me to push model performance beyond what was originally reported, while also focusing on metrics like false negatives, which are critical in fraud detection.

What Makes These Models Special?

Benefits of Each Model:

Logistic Reg

- Easy to implement and interpret.
- Works well for linearly separable data.
- Provides probabilities for class predictions.

LightGBM

- Very fast and memory efficient.
- Handles large datasets with high dimensionality well.
- Can handle categorical features natively.

XGBoost

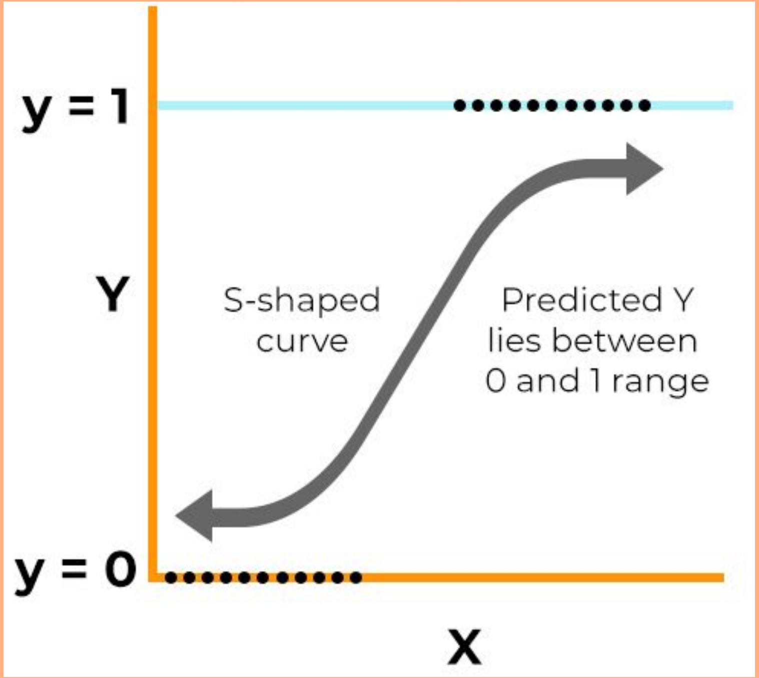
- Extremely fast and scalable for large datasets.
- Handles missing values automatically.
- High accuracy in many types of prediction problems.

Logistic Regression

$$y = \frac{e^{(b_0 + b_1 X)}}{1 + e^{(b_0 + b_1 X)}}$$

How it works:

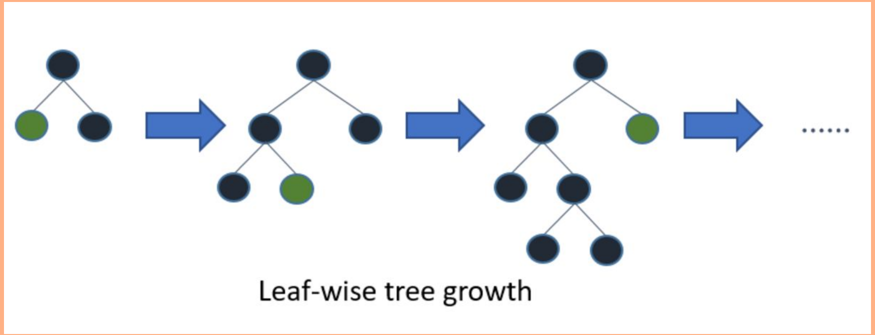
- A linear model that calculates the probability of a binary class (real or fake).
- It uses a sigmoid function to transform input features into a value between 0 and 1.
- Great for linearly separable data and interpretable results.



LightGBM

How it works:

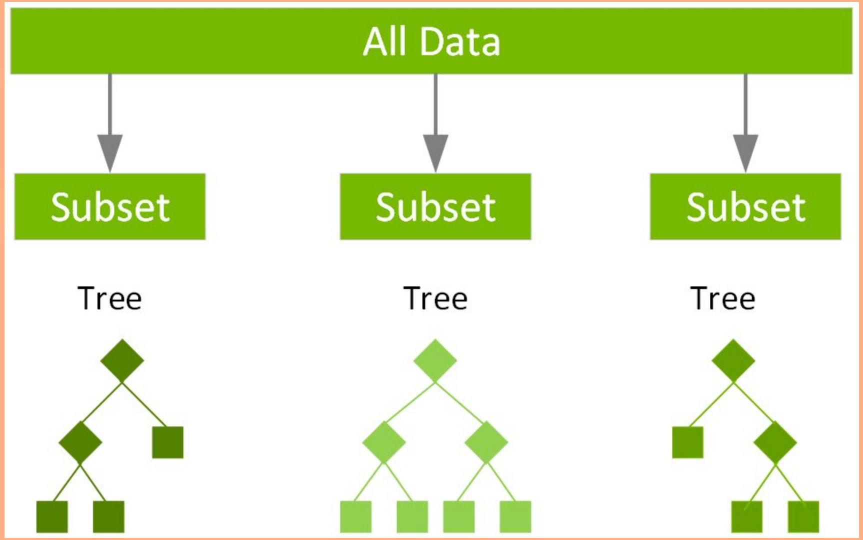
- A tree based gradient boosting algorithm.
- Trains models on small subsets of data, leading to faster performance.
- Handles categorical data and large datasets well.



XGBoost

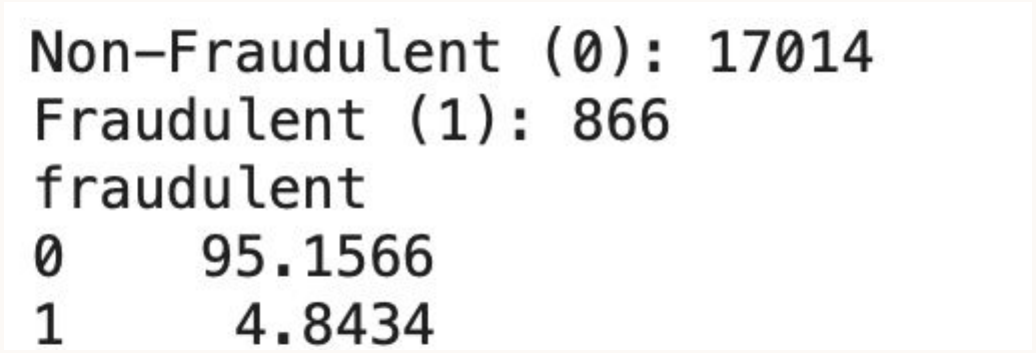
How it works:

- An optimized version of gradient boosting with built-in regularization.
- Prevents overfitting while maintaining high predictive power.
- Efficient in computation and supports parallel processing.



Dataset - Fake & Real Job postings

- ➔ Has 17 features and 17,880 rows of job postings.
- ➔ Since this is real data, it significantly unbalanced because there much more real jobs than fake.
- ➔ So there is only less than 5% of job postings that are fraudulent.



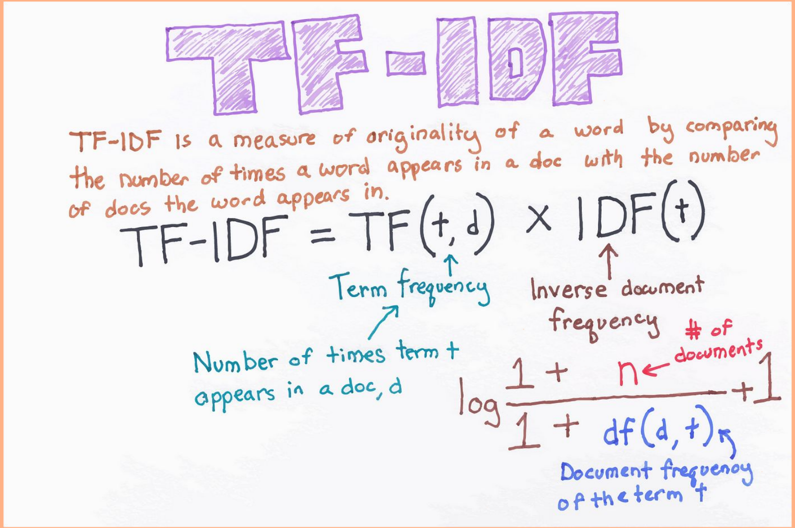
```
Non-Fraudulent (0): 17014
Fraudulent (1): 866
fraudulent
0    95.1566
1     4.8434
```

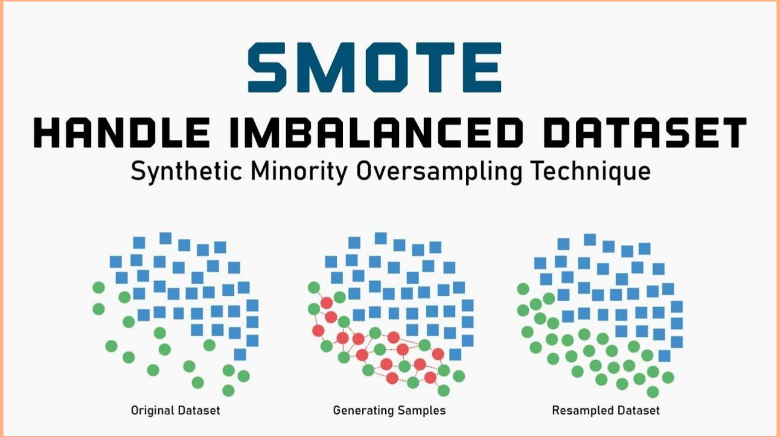
TF-IDF & SMOTE

How it works:

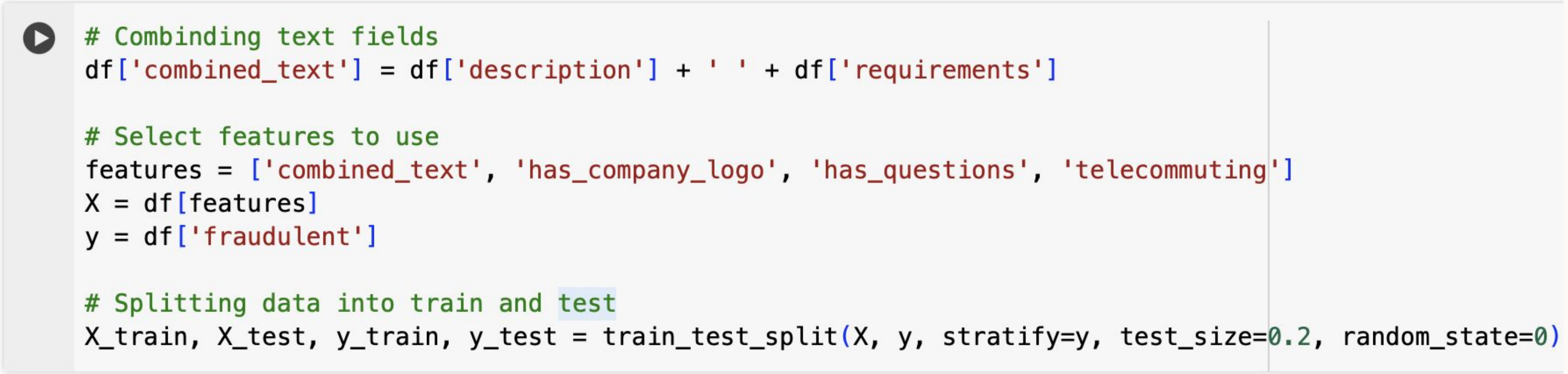
- TF-IDF: Converts text data into numerical values based on word importance. Reduces weight for common words and boosts rare but important ones.

- SMOTE: Oversamples the minority class by generating synthetic examples. Helps fix class imbalance so the model doesn't favor the majority (real) class.

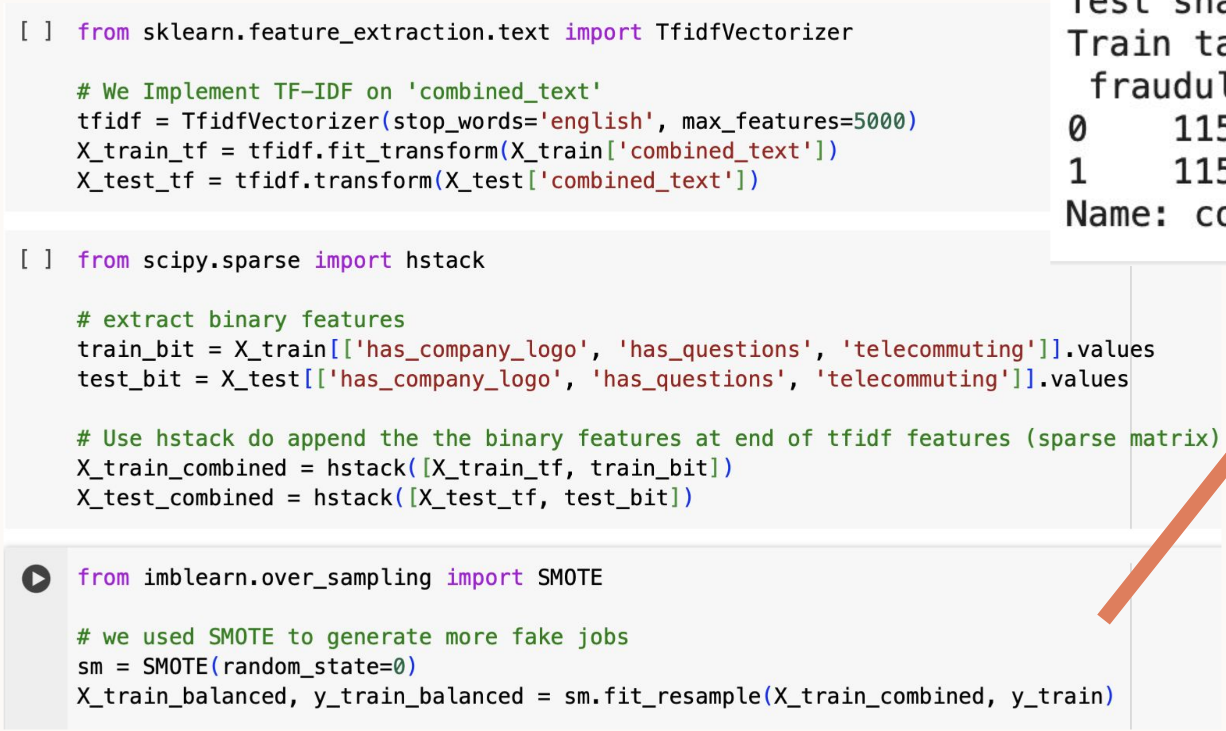

TF-IDF
TF-IDF is a measure of originality of a word by comparing the number of times a word appears in a doc with the number of docs the word appears in.
TF-IDF = TF(t, d) × IDF(t)
Term frequency
Inverse document frequency
Number of times term t appears in a doc, d
log (1 + n) / (1 + df(d,t)) + 1
# of documents
Document frequency of the term t


SMOTE
HANDLE IMBALANCED DATASET
Synthetic Minority Oversampling Technique
Original Dataset
Generating Samples
Resampled Dataset

Code - Data Preprocessing


```
# Combinding text fields
df['combined_text'] = df['description'] + ' ' + df['requirements']

# Select features to use
features = ['combined_text', 'has_company_logo', 'has_questions', 'telecommuting']
X = df[features]
y = df['fraudulent']

# Splitting data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=0)
```

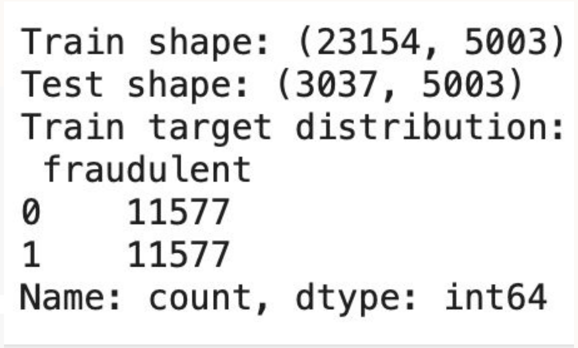
Code - TF-IDF & SMOTE



```python
from sklearn.feature_extraction.text import TfidfVectorizer

# We Implement TF-IDF on 'combined_text'
tfidf = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_tf = tfidf.fit_transform(X_train['combined_text'])
X_test_tf = tfidf.transform(X_test['combined_text'])
```

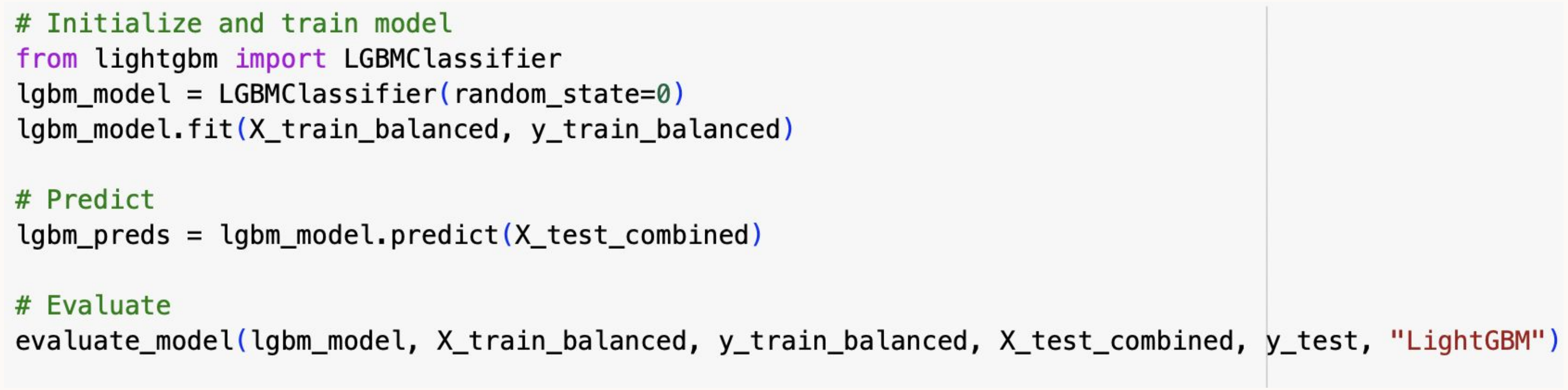
```python
from scipy.sparse import hstack

# extract binary features
train_bit = X_train[['has_company_logo', 'has_questions', 'telecommuting']].values
test_bit = X_test[['has_company_logo', 'has_questions', 'telecommuting']].values

# Use hstack do append the the binary features at end of tfidf features (sparse matrix)
X_train_combined = hstack([X_train_tf, train_bit])
X_test_combined = hstack([X_test_tf, test_bit])
```

```python
from imblearn.over_sampling import SMOTE

# we used SMOTE to generate more fake jobs
sm = SMOTE(random_state=0)
X_train_balanced, y_train_balanced = sm.fit_resample(X_train_combined, y_train)
```



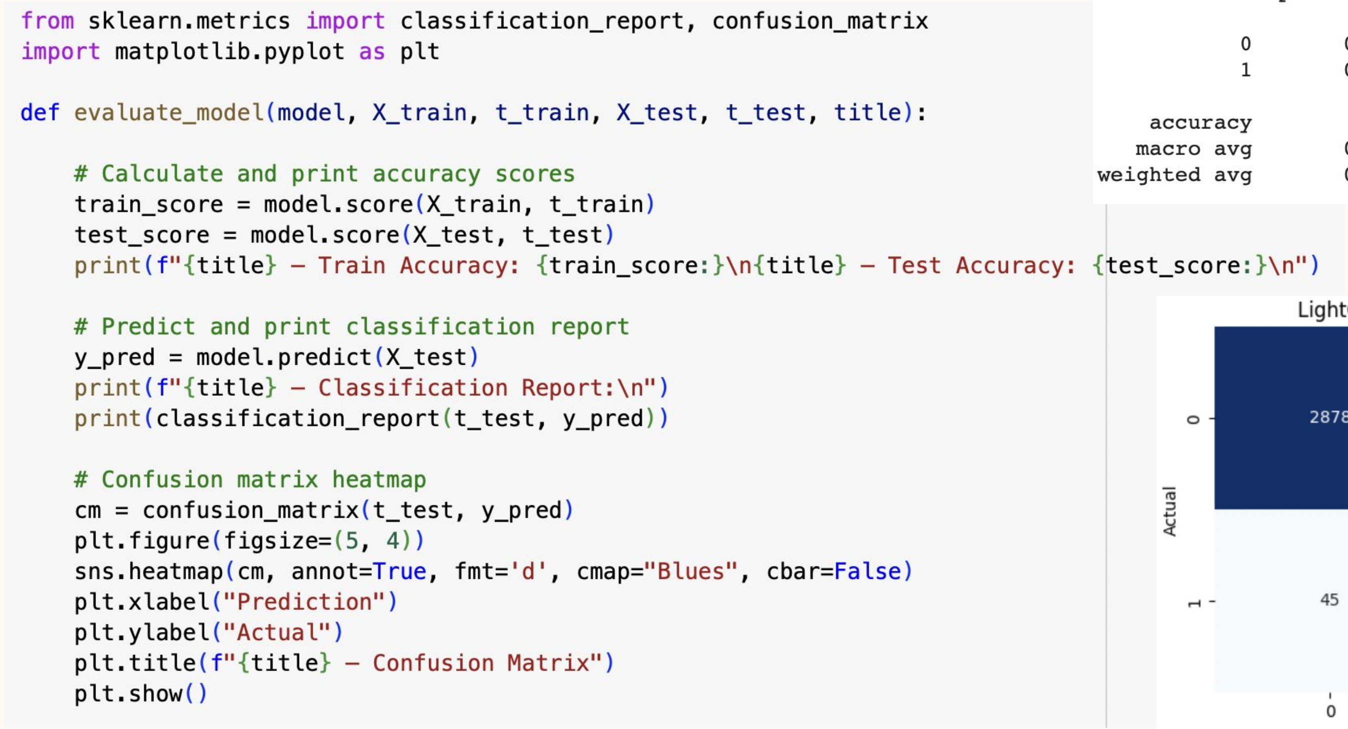
```
Train shape: (23154, 5003)
Test shape: (3037, 5003)
Train target distribution:
 fraudulent
0    11577
1    11577
Name: count, dtype: int64
```

Code - Baseline Model example (LightGBM)

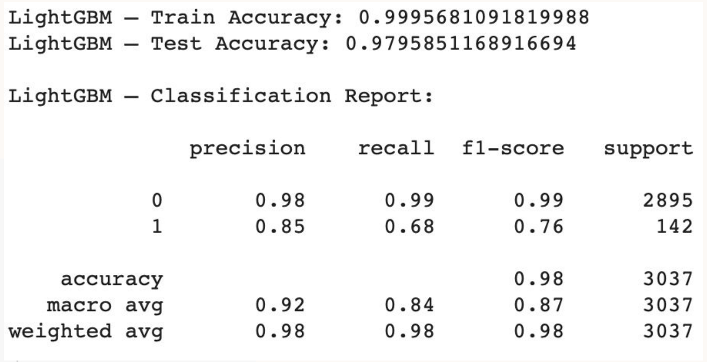


```python
# Initialize and train model
from lightgbm import LGBMClassifier
lgbm_model = LGBMClassifier(random_state=0)
lgbm_model.fit(X_train_balanced, y_train_balanced)

# Predict
lgbm_preds = lgbm_model.predict(X_test_combined)

# Evaluate
evaluate_model(lgbm_model, X_train_balanced, y_train_balanced, X_test_combined, y_test, "LightGBM")
```

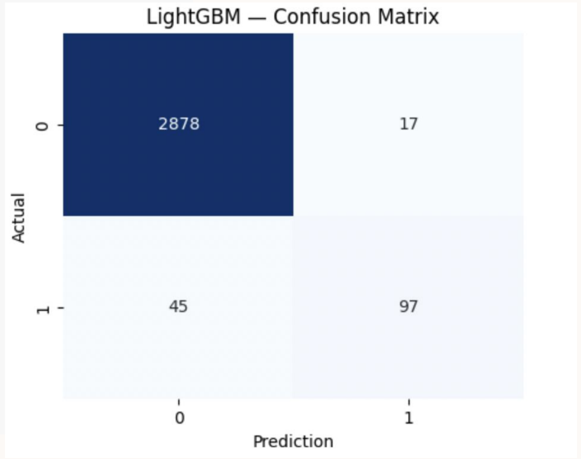
Code - Evaluation Metrics



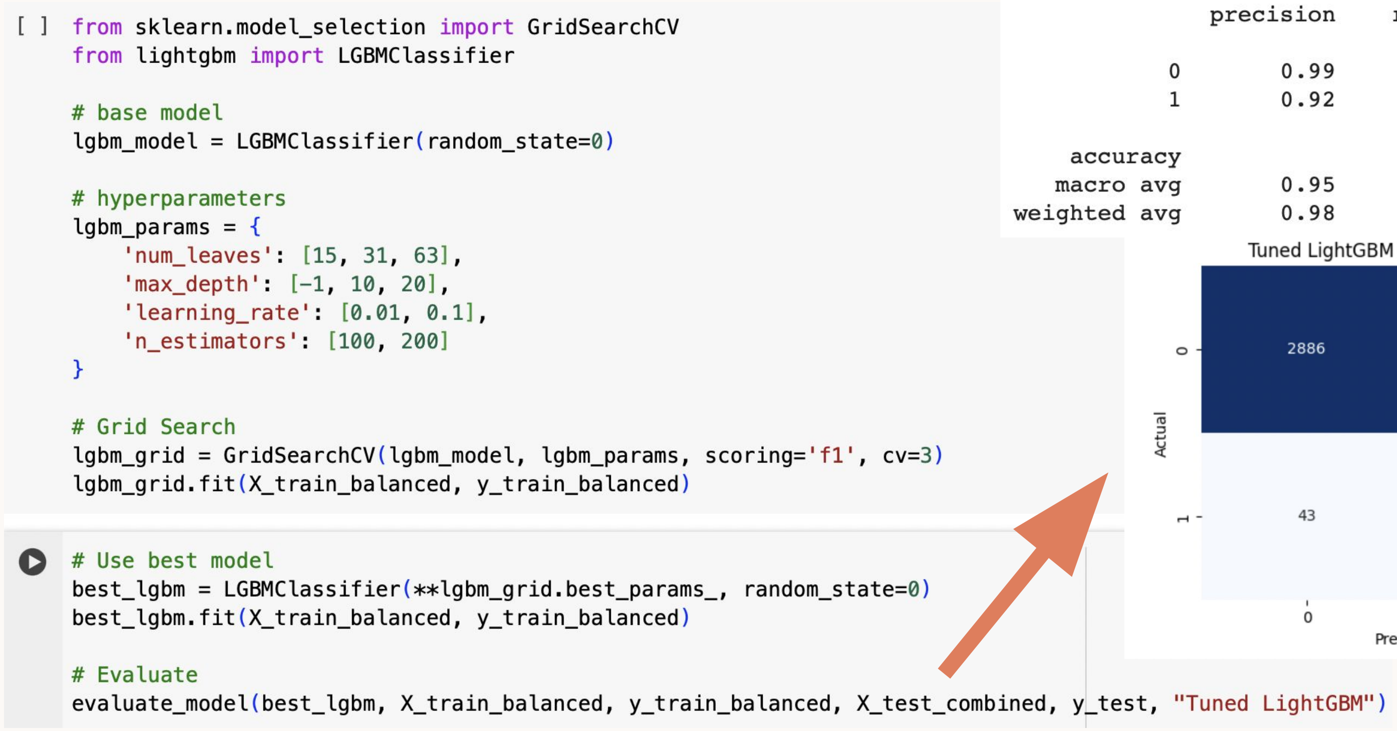
```python
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

def evaluate_model(model, X_train, t_train, X_test, t_test, title):

    # Calculate and print accuracy scores
    train_score = model.score(X_train, t_train)
    test_score = model.score(X_test, t_test)
    print(f"{title} — Train Accuracy: {train_score:}\n{title} — Test Accuracy: {test_score:}\n")

    # Predict and print classification report
    y_pred = model.predict(X_test)
    print(f"{title} — Classification Report:\n")
    print(classification_report(t_test, y_pred))

    # Confusion matrix heatmap
    cm = confusion_matrix(t_test, y_pred)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", cbar=False)
    plt.xlabel("Prediction")
    plt.ylabel("Actual")
    plt.title(f"{title} — Confusion Matrix")
    plt.show()
```



```
LightGBM — Train Accuracy: 0.9995681091819988
LightGBM — Test Accuracy: 0.9795851168916694

LightGBM — Classification Report:

              precision    recall  f1-score   support

           0       0.98      0.99      0.99      2895
           1       0.85      0.68      0.76       142

    accuracy                           0.98      3037
   macro avg       0.92      0.84      0.87      3037
weighted avg       0.98      0.98      0.98      3037
```

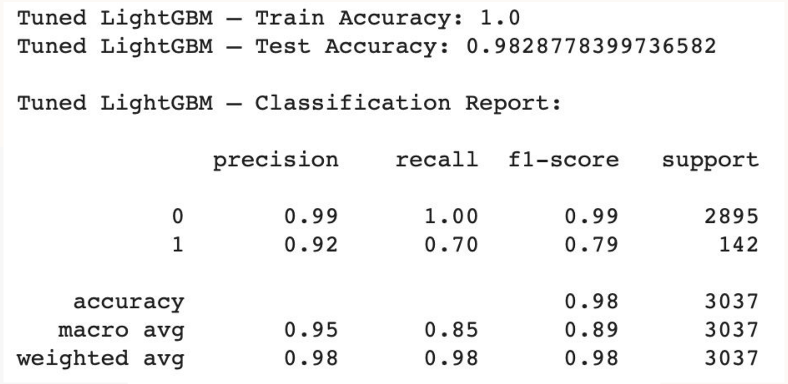


Code - GridSearchCV

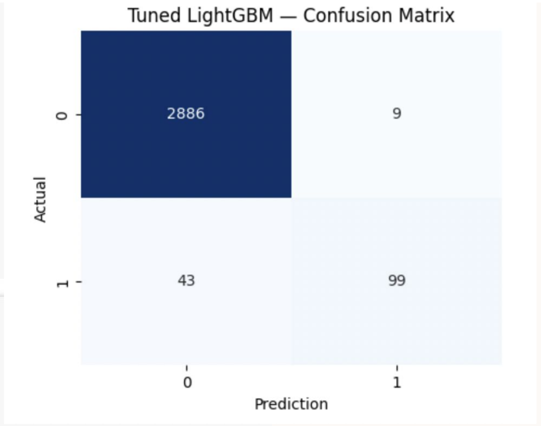


```
[ ] from sklearn.model_selection import GridSearchCV
    from lightgbm import LGBMClassifier

    # base model
    lgbm_model = LGBMClassifier(random_state=0)

    # hyperparameters
    lgbm_params = {
        'num_leaves': [15, 31, 63],
        'max_depth': [-1, 10, 20],
        'learning_rate': [0.01, 0.1],
        'n_estimators': [100, 200]
    }

    # Grid Search
    lgbm_grid = GridSearchCV(lgbm_model, lgbm_params, scoring='f1', cv=3)
    lgbm_grid.fit(X_train_balanced, y_train_balanced)
```

```
    # Use best model
    best_lgbm = LGBMClassifier(**lgbm_grid.best_params_, random_state=0)
    best_lgbm.fit(X_train_balanced, y_train_balanced)

    # Evaluate
    evaluate_model(best_lgbm, X_train_balanced, y_train_balanced, X_test_combined, y_test, "Tuned LightGBM")
```



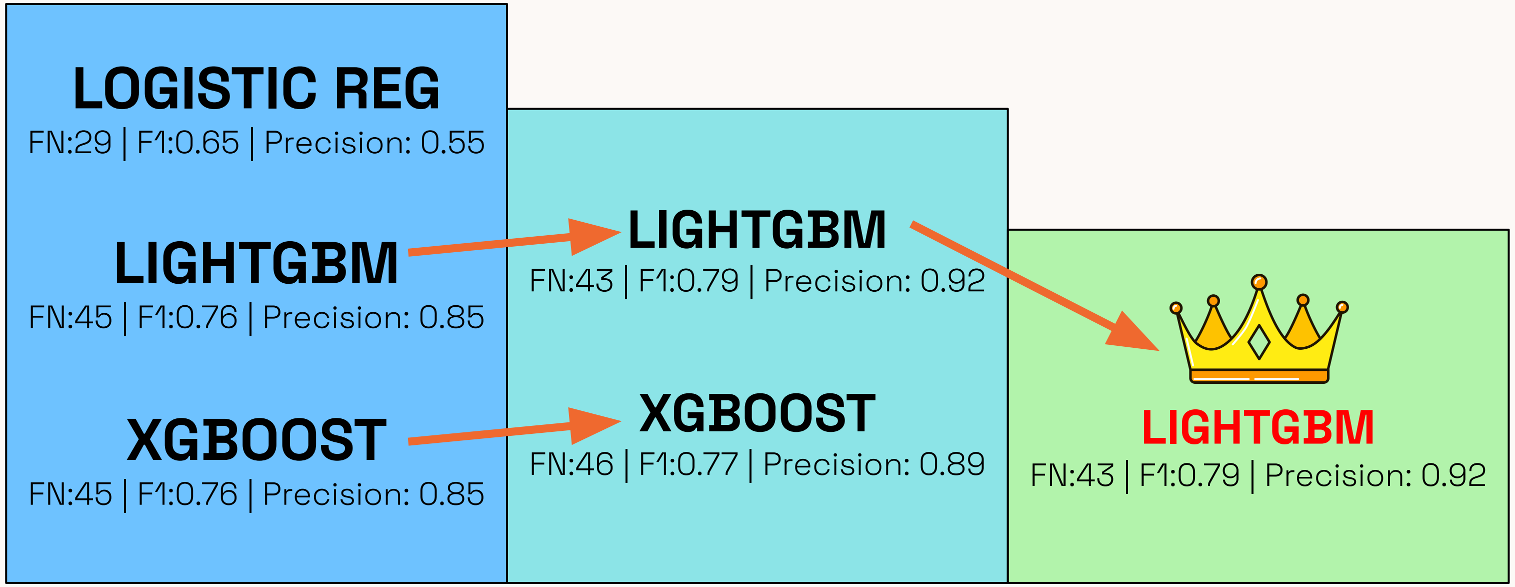
```
Tuned LightGBM — Train Accuracy: 1.0
Tuned LightGBM — Test Accuracy: 0.9828778399736582

Tuned LightGBM — Classification Report:

              precision    recall  f1-score   support

           0       0.99      1.00      0.99      2895
           1       0.92      0.70      0.79       142

    accuracy                           0.98      3037
   macro avg       0.95      0.85      0.89      3037
weighted avg       0.98      0.98      0.98      3037
```



04

Results & Observations

Model Pipeline Results


LOGISTIC REG
FN:29 | F1:0.65 | Precision: 0.55
LIGHTGBM
FN:45 | F1:0.76 | Precision: 0.85
XGBOOST
FN:45 | F1:0.76 | Precision: 0.85
LIGHTGBM
FN:43 | F1:0.79 | Precision: 0.92
XGBOOST
FN:46 | F1:0.77 | Precision: 0.89
LIGHTGBM
FN:43 | F1:0.79 | Precision: 0.92

Observations

In fraud detection for job listings, both precision and recall matter. While catching fraud is important, falsely labeling legitimate jobs could discourage users and harm platform trust. Therefore, models like LightGBM and XGBoost were chosen because they offered a strong balance between catching fraud and avoiding false alarms, reflected in their higher F1-scores.

05

Conclusion & Future Work

Conclusion

Project results

After testing several models, LightGBM and XGBoost offered the best balance between precision and recall. The combination of text features (TF-IDF), class balancing (SMOTE), and hyperparameter tuning (GridSearchCV) led to strong model performance. This POC shows potential for deployment in job platforms to flag potential scams. Using GridSearchCV with these models took multiple hours to run but returned greatly improved results.

Future Work

Although, this project is currently only binary classification, it can be turned into a multi classification if the we planned to detect job postings as Real, Suspicious, and Fraudulent. This gives applicants a chance to review and make their own decision on whether they think the job posting seems worth applying.

Finally, a more complex and difficult implementation is to attempt to detect “ghost jobs”. Which is when a company post jobs where they have no intention to hire. They do this to increase work productivity of current employees and boost revenue sales by giving the impression the company is expanding.

Citations

https://blissiree.com/10-best-mindfulness-tips-for-stressed-out-men/

https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-logistic-regression/

https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

https://www.nvidia.com/en-us/glossary/xgboost/

https://www.kaggle.com/datasets/shivamb/real-or-fake-fake-jobposting-prediction

https://creazilla.com/media/clipart/79696/princess-crown

https://www.shutterstock.com/image-photo/business-graphs-charts-magnifying-glass-on-2041846232

https://www.datakwery.com/post/approaching-data-analysis/

https://medium.com/analytics-vidhya/demonstrating-calculation-of-tf-idf-from-sklearn-4f9526e7e78b

https://python.plainenglish.io/balancing-the-scales-how-smote-transforms-machine-learning-with-imbalanced-data-a6d3367254cd