

Assignment 2 – NLP Classification

Arabic HRSD Question Classification Using Machine Learning

This project applies Natural Language Processing (NLP) techniques to classify Arabic questions from the Human Resources and Social Development (HRSD) FAQ dataset into three main sectors:

- **Labor Sector**
- **Civil Service Sector**
- **Social Development Sector**

The project includes the following steps:

1. Dataset loading and inspection
2. Data cleaning and duplicate removal
3. Arabic text preprocessing
4. Train-test split using stratified sampling
5. Feature representation using:
 - Bag of Words (BoW)
 - TF-IDF with Unigrams and Bigrams
6. Handling class imbalance using Random Oversampling
7. Training multiple machine learning classifiers:
 - Logistic Regression
 - Support Vector Machine (SVM)
 - Random Forest
 - Decision Tree
8. Model evaluation using:
 - Accuracy
 - Precision
 - Recall
 - F1 Score
 - Confusion Matrix
9. Final model comparison and selection of the best-performing model

Best Model

The best-performing configuration was:

Support Vector Machine (SVM) + TF-IDF (Unigrams and Bigrams) + Random Oversampling

with:

- **Accuracy: 90.91%**
- **Precision: 91.71%**
- **Recall: 90.91%**
- **F1 Score: 90.72%**

In [1]: `%pip install pandas numpy scikit-learn matplotlib nltk`

```
Requirement already satisfied: pandas in .\Lib\site-packages (3.0.5)
Requirement already satisfied: numpy in .\Lib\site-packages (2.5.2)
Requirement already satisfied: scikit-learn in .\Lib\site-packages (1.9.0)
Requirement already satisfied: matplotlib in .\Lib\site-packages (3.11.1)
Requirement already satisfied: nltk in .\Lib\site-packages (3.10.3)
Requirement already satisfied: python-dateutil>=2.8.2 in .\Lib\site-packages (from p
andas) (2.9.0.post0)
Requirement already satisfied: tzdata in .\Lib\site-packages (from pandas) (2026.3)
Requirement already satisfied: scipy>=1.10.0 in .\Lib\site-packages (from scikit-lea
rn) (1.18.0)
Requirement already satisfied: joblib>=1.4.0 in .\Lib\site-packages (from scikit-lea
rn) (1.5.3)
Requirement already satisfied: narwhals>=2.0.1 in .\Lib\site-packages (from scikit-l
earn) (2.25.0)
Requirement already satisfied: threadpoolctl>=3.5.0 in .\Lib\site-packages (from sci
kit-learn) (3.6.0)
Requirement already satisfied: contourpy>=1.0.1 in .\Lib\site-packages (from matplot
lib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in .\Lib\site-packages (from matplotlib)
(0.12.1)
Requirement already satisfied: fonttools>=4.28.2 in .\Lib\site-packages (from matplo
tlib) (4.63.0)
Requirement already satisfied: kiwisolver>=1.3.1 in .\Lib\site-packages (from matplo
tlib) (1.5.0)
Requirement already satisfied: packaging>=20.0 in .\Lib\site-packages (from matplotl
ib) (26.2)
Requirement already satisfied: pillow>=9 in .\Lib\site-packages (from matplotlib) (1
2.3.0)
Requirement already satisfied: pyparsing>=3 in .\Lib\site-packages (from matplotlib)
(3.3.2)
Requirement already satisfied: defusedxml in .\Lib\site-packages (from nltk) (0.7.1)
Requirement already satisfied: click in .\Lib\site-packages (from nltk) (8.4.2)
Requirement already satisfied: regex>=2021.8.3 in .\Lib\site-packages (from nltk) (2
026.7.19)
Requirement already satisfied: tqdm in .\Lib\site-packages (from nltk) (4.70.0)
Requirement already satisfied: six>=1.5 in .\Lib\site-packages (from python-dateutil
>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: colorama in .\Lib\site-packages (from click->nltk) (0
.4.6)
Note: you may need to restart the kernel to use updated packages.
```

In [2]: `# =====`
`# Cell 2 - Load the Dataset`
`# =====`

```
import pandas as pd

file_path = "hrsd_faq_clean.csv"

df = pd.read_csv(file_path)

print("Dataset loaded successfully.")
print("Dataset shape:", df.shape)

print("\nColumns:")
print(df.columns.tolist())

print("\nFirst 5 rows:")
display(df.head())
```

Dataset loaded successfully.

Dataset shape: (711, 10)

Columns:

['faq_id', 'question', 'answer', 'beneficiaries', 'sector', 'subsite', 'page_number', 'page_url', 'source', 'searchable_text']

First 5 rows:

faq_id	question	answer	beneficiaries	sector	subsite	page_number	
0	1	هل تستخدم البوابة والانظمة التابعة لها ... الحلول	تقدم الوزارة خدمة التوقيع الالكتروني في اطار ...	أصحاب عمل أفراد العمالة المنزلية كبار السن	قطاع العمل	مركز الرياض للسياسات السلوكية	1 https://www.hr
1	2	ماذا أفعل إذا تم إرجاع الطلب للمراجعة؟	الدخول إلى "مهامي" ← اختيار مهمة طلب ← مراجعة ...	التخصصات الاجتماعية العمالة المنزلية كبار ...	قطاع التنمية الاجتماعية	مركز الرياض للسياسات السلوكية	1 https://www.hr
2	3	ماذا يحدث بعد إرسال الطلب؟	يتم تحويل الطلب إلى اللجنة للمراجعة ويتم إشعار ...	التخصصات الاجتماعية العمالة المنزلية كبار ...	قطاع التنمية الاجتماعية	مركز الرياض للسياسات السلوكية	1 https://www.hr
3	4	كيف أقدم طلب تعديل فئة مهنية؟	من خلال تسجيل الدخول ← الخدمات ← اختيار "طلب" ...	التخصصات الاجتماعية العمالة المنزلية كبار ...	قطاع التنمية الاجتماعية	مركز الرياض للسياسات السلوكية	1 https://www.hr
4	5	ما متطلبات التقديم؟	أن يكون المتقدم حاصل على ترخيص سابق.	التخصصات الاجتماعية العمالة المنزلية كبار ...	قطاع التنمية الاجتماعية	مركز الرياض للسياسات السلوكية	1 https://www.hr

```

In [3]: # =====
# Cell 3 - Dataset Inspection
# =====

print("Dataset shape:", df.shape)

# Check missing values
print("\nMissing values per column:")
print(df.isnull().sum())

# Check sector distribution

```

```

print("\nSector distribution:")
print(df['sector'].value_counts(dropna=False))

# Check duplicated questions
duplicate_questions = df.duplicated(subset=['question']).sum()
print("\nNumber of duplicated questions:", duplicate_questions)

# Show duplicated questions if any
if duplicate_questions > 0:
    display(
        df[df.duplicated(subset=['question'], keep=False)]
        .sort_values('question')
        [['faq_id', 'question', 'sector']]
    )

```

Dataset shape: (711, 10)

Missing values per column:

faq_id	0
question	0
answer	0
beneficiaries	0
sector	87
subsite	0
page_number	0
page_url	0
source	0
searchable_text	0

dtype: int64

Sector distribution:

sector	
275	قطاع العمل
230	قطاع الخدمة المدنية (العام)
105	قطاع التنمية الاجتماعية
NaN	
10	قطاع التنمية الاجتماعية قطاع الخدمة المدنية (العام) قطاع العمل
4	قطاع التنمية الاجتماعية قطاع العمل

Name: count, dtype: int64

Number of duplicated questions: 8

	faq_id	question	sector
686	687	كيف يمكن تقديم طلب الخدمة؟	قطاع التنمية الاجتماعية
691	692	كيف يمكن تقديم طلب الخدمة؟	قطاع التنمية الاجتماعية
30	31	كيف يمكن متابعة حالة الطلب؟	قطاع التنمية الاجتماعية
454	455	كيف يمكن متابعة حالة الطلب؟	قطاع التنمية الاجتماعية
690	691	كيف يمكن متابعة حالة الطلب؟	قطاع التنمية الاجتماعية
705	706	كيف يمكن متابعة حالة الطلب؟	قطاع التنمية الاجتماعية
91	92	ما هو نظام حماية الأجور؟	NaN
248	249	ما هو نظام حماية الأجور؟	قطاع العمل
93	94	ما هي أهداف برنامج حماية الأجور؟	NaN
247	248	ما هي أهداف برنامج حماية الأجور؟	قطاع العمل
687	688	ماهي شروط تقديم الطلب؟	قطاع التنمية الاجتماعية
692	693	ماهي شروط تقديم الطلب؟	قطاع التنمية الاجتماعية
90	91	...هل أستطيع رفع ملفات حماية الأجور عبر موقع وزار	NaN
246	247	...هل أستطيع رفع ملفات حماية الأجور عبر موقع وزار	قطاع العمل

```
In [4]: # =====
# Cell 4 - Select Main Classes
# =====

# Define the three target classes
main_sectors = [
    'قطاع العمل',
    'قطاع الخدمة المدنية (العام)',
    'قطاع التنمية الاجتماعية'
]

# Keep only rows belonging to one of the three main sectors
df_clean = df[df['sector'].isin(main_sectors)].copy()

print("Dataset size before filtering:", len(df))
print("Dataset size after selecting main sectors:", len(df_clean))

print("\nClass distribution before removing duplicates:")
print(df_clean['sector'].value_counts())

# Check duplicated questions inside the selected dataset
duplicates = df_clean[
    df_clean.duplicated(subset=['question'], keep=False)
].sort_values('question')

print("\nNumber of rows involved in duplicated questions:", len(duplicates))
```

```
if len(duplicates) > 0:
    display(duplicates[['faq_id', 'question', 'sector']])
```

Dataset size before filtering: 711

Dataset size after selecting main sectors: 610

Class distribution before removing duplicates:

sector

275 قطاع العمل

230 قطاع الخدمة المدنية (العام)

105 قطاع التنمية الاجتماعية

Name: count, dtype: int64

Number of rows involved in duplicated questions: 8

	faq_id	question	sector
686	687	كيف يمكن تقديم طلب الخدمة؟	قطاع التنمية الاجتماعية
691	692	كيف يمكن تقديم طلب الخدمة؟	قطاع التنمية الاجتماعية
30	31	كيف يمكن متابعة حالة الطلب؟	قطاع التنمية الاجتماعية
454	455	كيف يمكن متابعة حالة الطلب؟	قطاع التنمية الاجتماعية
690	691	كيف يمكن متابعة حالة الطلب؟	قطاع التنمية الاجتماعية
705	706	كيف يمكن متابعة حالة الطلب؟	قطاع التنمية الاجتماعية
687	688	ماهي شروط تقديم الطلب؟	قطاع التنمية الاجتماعية
692	693	ماهي شروط تقديم الطلب؟	قطاع التنمية الاجتماعية

```
In [5]: # =====
# Cell 5 - Remove Duplicates
# =====

# Remove duplicated questions and keep the first occurrence
df_clean = df_clean.drop_duplicates(
    subset=['question'],
    keep='first'
).copy()

# Reset index
df_clean = df_clean.reset_index(drop=True)

print("Dataset size after removing duplicates:", len(df_clean))

print("\nClass distribution after removing duplicates:")
print(df_clean['sector'].value_counts())

print("\nRemaining duplicated questions:",
      df_clean.duplicated(subset=['question']).sum())
```

Dataset size after removing duplicates: 605

Class distribution after removing duplicates:

sector

275 قطاع العمل

230 قطاع الخدمة المدنية (العام)

100 قطاع التنمية الاجتماعية

Name: count, dtype: int64

Remaining duplicated questions: 0

```
In [6]: # =====
# Cell 6 - Create Class Labels
# =====

label_map = {
    '0': 'قطاع العمل',
    '1': 'قطاع الخدمة المدنية (العام)',
    '2': 'قطاع التنمية الاجتماعية'
}

df_clean['label'] = df_clean['sector'].map(label_map)

print("Label distribution:")
print(df_clean['label'].value_counts().sort_index())

display(
    df_clean[['question', 'sector', 'label']].head(10)
)
```

Label distribution:

label

0 275

1 230

2 100

Name: count, dtype: int64

	question	sector	label
0	... هل تستخدم البوابة والانظمة التابعة لها الحلول	قطاع العمل	0
1	ماذا أفعل إذا تم إرجاع الطلب للمراجعة؟	قطاع التنمية الاجتماعية	2
2	ماذا يحدث بعد إرسال الطلب؟	قطاع التنمية الاجتماعية	2
3	كيف أقدم طلب تعديل فئة مهنية؟	قطاع التنمية الاجتماعية	2
4	ما متطلبات التقديم؟	قطاع التنمية الاجتماعية	2
5	...هل يمكن للمتقدم أن يحصل على أكثر من تصنيف وبال	قطاع التنمية الاجتماعية	2
6	...إذا رفض طلب التسجيل والتصنيف هل يحق لي استرداد	قطاع التنمية الاجتماعية	2
7	هل يمكن إلغاء الفاتورة؟	قطاع التنمية الاجتماعية	2
8	ماذا يحدث بعد قبول الطلب؟	قطاع التنمية الاجتماعية	2
9	ما هي حالات الطلب؟	قطاع التنمية الاجتماعية	2

```

In [7]: # =====
# Cell 7 - Arabic Text Preprocessing
# =====

import re

def preprocess_arabic_text(text):

    # Convert input to string
    text = str(text)

    # 1. Remove Arabic diacritics (Tashkeel)
    arabic_diacritics = re.compile(r"""
        | # Tashdid
        | # Fatha
        | # Tanwin Fath
        | # Damma
        | # Tanwin Damm
        | # Kasra
        | # Tanwin Kasr
        | # Sukun
        - # Tatweel
    """, re.VERBOSE)

    text = re.sub(arabic_diacritics, '', text)

    # 2. Normalize Arabic letters
    text = re.sub(r'[\u0600-\u06FF]', '', text)
    text = re.sub(r'ي', 'ى', text)

    # 3. Remove URLs
    text = re.sub(r'http\S+|www\.\S+', ' ', text)

    # 4. Remove English letters and numbers
    text = re.sub(r'[A-Za-z0-9]', ' ', text)

    # 5. Keep Arabic characters only
    text = re.sub(r'^[\u0600-\u06FF\s]', ' ', text)

    # 6. Remove Arabic punctuation
    text = re.sub(r'[:;«»]', ' ', text)

    # 7. Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()

    return text

# Apply preprocessing to the questions
df_clean['clean_question'] = df_clean['question'].apply(
    preprocess_arabic_text
)

print("Preprocessing completed successfully.")

```

```
# Display examples before and after preprocessing
display(
    df_clean[['question', 'clean_question']].head(10)
)
```

Preprocessing completed successfully.

	question	clean_question
0	... هل تستخدم البوابة والانظمة التابعة لها الحلول	... هل تستخدم البوابة والانظمة التابعة لها الحلول
1	ماذا أفعل إذا تم إرجاع الطلب للمراجعة؟	ماذا افعل اذا تم ارجاع الطلب للمراجعة
2	ماذا يحدث بعد إرسال الطلب؟	ماذا يحدث بعد ارسال الطلب
3	كيف أقدم طلب تعديل فئة مهنية؟	كيف اقدم طلب تعديل فئة مهنية
4	ما متطلبات التقديم؟	ما متطلبات التقديم
5	هل يمكن للمتقدم أن يحصل على أكثر من تصنيف...وبال	هل يمكن للمتقدم ان يحصل علي اكثر من تصنيف...وبال
6	إذا رفض طلب التسجيل والتصنيف هل يحق لي...استرداد	اذا رفض طلب التسجيل والتصنيف هل يحق لي...استرداد
7	هل يمكن إلغاء الفاتورة؟	هل يمكن الغاء الفاتورة
8	ماذا يحدث بعد قبول الطلب؟	ماذا يحدث بعد قبول الطلب
9	ما هي حالات الطلب؟	ما هي حالات الطلب

```
In [8]: # =====
# Cell 8 - Define Features and Target
# Train/Test Split
# =====

from sklearn.model_selection import train_test_split

X = df_clean['clean_question']
y = df_clean['label']

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.20,
    random_state=42,
    stratify=y
)

print("Training samples:", len(X_train))
print("Testing samples:", len(X_test))

print("\nTraining class distribution:")
print(y_train.value_counts().sort_index())

print("\nTesting class distribution:")
print(y_test.value_counts().sort_index())
```

Training samples: 484

Testing samples: 121

Training class distribution:

label

0 220

1 184

2 80

Name: count, dtype: int64

Testing class distribution:

label

0 55

1 46

2 20

Name: count, dtype: int64

```
In [9]: # =====  
# Cell 9 - Feature Representation  
# Bag of Words  
# =====  
  
from sklearn.feature_extraction.text import CountVectorizer  
  
bow_vectorizer = CountVectorizer()  
  
# Fit only on training data  
X_train_bow = bow_vectorizer.fit_transform(X_train)  
  
# Transform test data using the same vocabulary  
X_test_bow = bow_vectorizer.transform(X_test)  
  
print("Bag of Words created successfully.")  
print("Training matrix shape:", X_train_bow.shape)  
print("Testing matrix shape:", X_test_bow.shape)  
print("Vocabulary size:", len(bow_vectorizer.vocabulary_))
```

Bag of Words created successfully.

Training matrix shape: (484, 1922)

Testing matrix shape: (121, 1922)

Vocabulary size: 1922

```
In [10]: %pip install imbalanced-learn
```

Requirement already satisfied: imbalanced-learn in .\Lib\site-packages (0.14.2)
 Requirement already satisfied: numpy<3,>=1.25.2 in .\Lib\site-packages (from imbalanced-learn) (2.5.2)
 Requirement already satisfied: scipy<2,>=1.11.4 in .\Lib\site-packages (from imbalanced-learn) (1.18.0)
 Requirement already satisfied: scikit-learn<2,>=1.4.2 in .\Lib\site-packages (from imbalanced-learn) (1.9.0)
 Requirement already satisfied: sklearn-compat<0.2,>=0.1.6 in .\Lib\site-packages (from imbalanced-learn) (0.1.6)
 Requirement already satisfied: joblib<2,>=1.2.0 in .\Lib\site-packages (from imbalanced-learn) (1.5.3)
 Requirement already satisfied: threadpoolctl<4,>=2.0.0 in .\Lib\site-packages (from imbalanced-learn) (3.6.0)
 Requirement already satisfied: narwhals>=2.0.1 in .\Lib\site-packages (from scikit-learn<2,>=1.4.2->imbalanced-learn) (2.25.0)
 Note: you may need to restart the kernel to use updated packages.

```
In [11]: # =====
# Logistic Regression
# Bag of Words - Without Oversampling
# =====

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix,
    ConfusionMatrixDisplay
)

import matplotlib.pyplot as plt

# Create model
lr_bow = LogisticRegression(
    max_iter=1000,
    random_state=42
)

# Train on original training data
lr_bow.fit(
    X_train_bow,
    y_train
)

# Predict on original test data
y_pred_lr_bow = lr_bow.predict(X_test_bow)

# Evaluation
accuracy = accuracy_score(
    y_test,
    y_pred_lr_bow
)
```

```
precision = precision_score(
    y_test,
    y_pred_lr_bow,
    average='weighted',
    zero_division=0
)

recall = recall_score(
    y_test,
    y_pred_lr_bow,
    average='weighted',
    zero_division=0
)

f1 = f1_score(
    y_test,
    y_pred_lr_bow,
    average='weighted',
    zero_division=0
)

print("Logistic Regression - Bag of Words")
print("-----")
print(f"Accuracy : {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall   : {recall:.4f}")
print(f"F1 Score : {f1:.4f}")

print("\nClassification Report:")
print(
    classification_report(
        y_test,
        y_pred_lr_bow,
        target_names=[
            "Labor Sector",
            "Civil Service Sector",
            "Social Development Sector"
        ],
        zero_division=0
    )
)

cm_bow = confusion_matrix(
    y_test,
    y_pred_lr_bow
)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm_bow,
    display_labels=[
        "Labor",
        "Civil Service",
        "Social Development"
    ]
)
```

```
disp.plot()
plt.title("Logistic Regression - Bag of Words")
plt.show()
```

Logistic Regression - Bag of Words

Accuracy : 0.8595

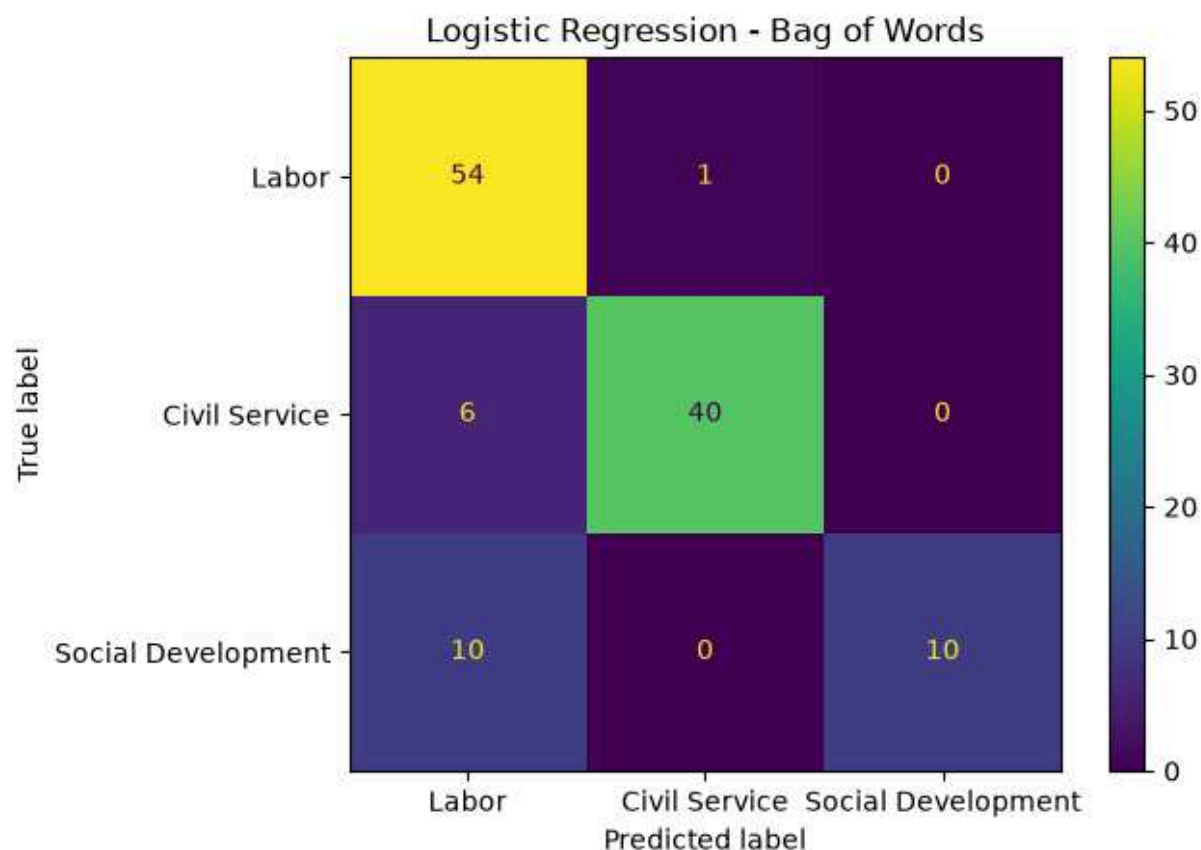
Precision: 0.8868

Recall : 0.8595

F1 Score : 0.8525

Classification Report:

	precision	recall	f1-score	support
Labor Sector	0.77	0.98	0.86	55
Civil Service Sector	0.98	0.87	0.92	46
Social Development Sector	1.00	0.50	0.67	20
accuracy			0.86	121
macro avg	0.92	0.78	0.82	121
weighted avg	0.89	0.86	0.85	121



```
In [12]: # =====
# Cell 10 - Handle Class Imbalance
# Random Oversampling
# =====

from imblearn.over_sampling import RandomOverSampler
from collections import Counter
```

```

print("Before Oversampling:")
print(Counter(y_train))

ros = RandomOverSampler(random_state=42)

X_train_bow_balanced, y_train_balanced = ros.fit_resample(
    X_train_bow,
    y_train
)

print("\nAfter Oversampling:")
print(Counter(y_train_balanced))

print("\nBalanced training shape:")
print(X_train_bow_balanced.shape)

```

Before Oversampling:

Counter({0: 220, 1: 184, 2: 80})

After Oversampling:

Counter({0: 220, 2: 220, 1: 220})

Balanced training shape:

(660, 1922)

```

In [13]: # =====
# Cell 11 - Logistic Regression
# Bag of Words + Random Oversampling
# =====

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix,
    ConfusionMatrixDisplay
)

import matplotlib.pyplot as plt

# Create model
lr_bow_ros = LogisticRegression(
    max_iter=1000,
    random_state=42
)

# Train on balanced training data
lr_bow_ros.fit(
    X_train_bow_balanced,
    y_train_balanced
)

```

```

# Predict on ORIGINAL test data
y_pred_lr_bow_ros = lr_bow_ros.predict(X_test_bow)

# Evaluation
accuracy_ros = accuracy_score(y_test, y_pred_lr_bow_ros)

precision_ros = precision_score(
    y_test,
    y_pred_lr_bow_ros,
    average='weighted',
    zero_division=0
)

recall_ros = recall_score(
    y_test,
    y_pred_lr_bow_ros,
    average='weighted',
    zero_division=0
)

f1_ros = f1_score(
    y_test,
    y_pred_lr_bow_ros,
    average='weighted',
    zero_division=0
)

print("Logistic Regression - Bag of Words + Random Oversampling")
print("-----")
print(f"Accuracy : {accuracy_ros:.4f}")
print(f"Precision: {precision_ros:.4f}")
print(f"Recall   : {recall_ros:.4f}")
print(f"F1 Score : {f1_ros:.4f}")

print("\nClassification Report:")
print(
    classification_report(
        y_test,
        y_pred_lr_bow_ros,
        target_names=[
            "Labor Sector",
            "Civil Service Sector",
            "Social Development Sector"
        ],
        zero_division=0
    )
)

# Confusion Matrix
cm_ros = confusion_matrix(y_test, y_pred_lr_bow_ros)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm_ros,
    display_labels=[
        "Labor",
        "Civil Service",

```

```

        "Social Development"
    ]
)

disp.plot()
plt.title("Logistic Regression - BoW + Oversampling")
plt.show()

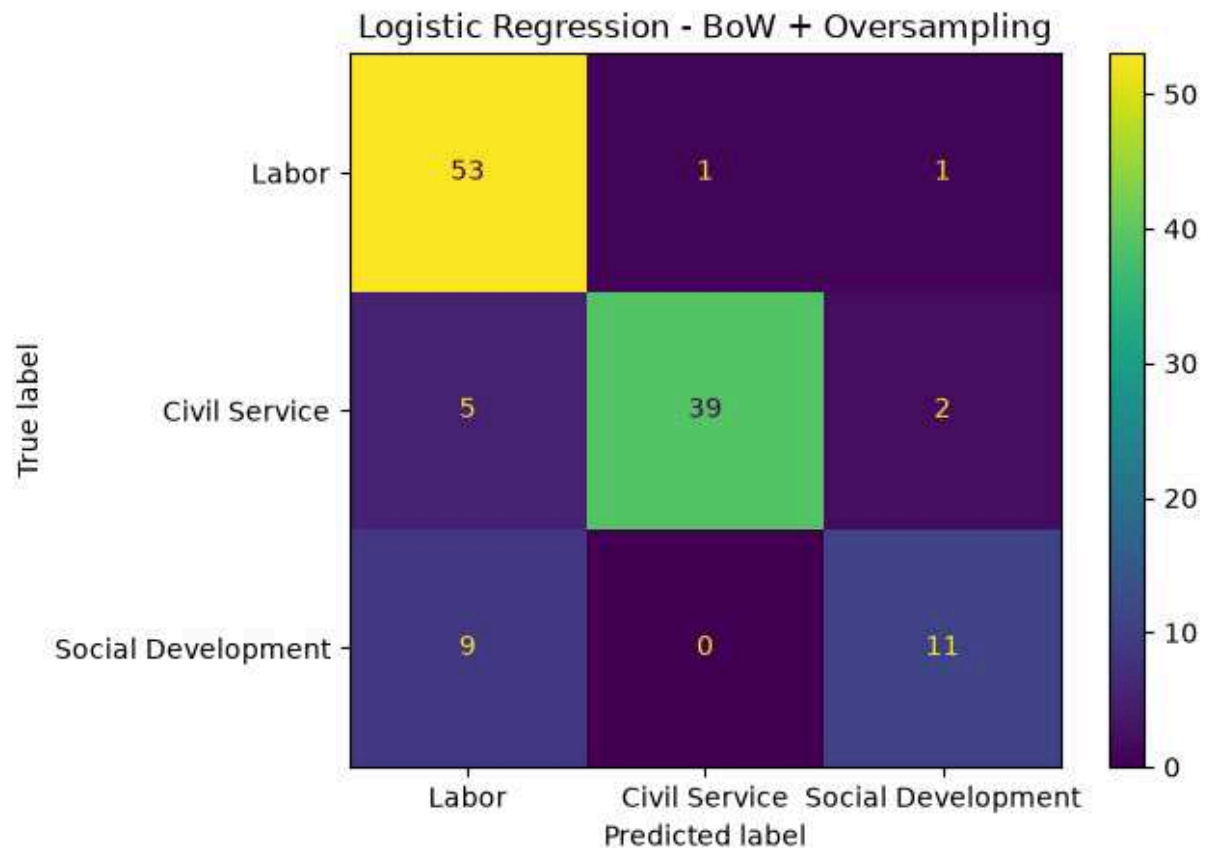
```

Logistic Regression - Bag of Words + Random Oversampling

 Accuracy : 0.8512
 Precision: 0.8601
 Recall : 0.8512
 F1 Score : 0.8467

Classification Report:

	precision	recall	f1-score	support
Labor Sector	0.79	0.96	0.87	55
Civil Service Sector	0.97	0.85	0.91	46
Social Development Sector	0.79	0.55	0.65	20
accuracy			0.85	121
macro avg	0.85	0.79	0.81	121
weighted avg	0.86	0.85	0.85	121



```

In [14]: # =====
# Cell 11 - Store Model Results
# =====

```

```

results = []

results.append({
    'Feature Representation': 'Bag of Words',
    'Model': 'Logistic Regression',
    'Accuracy': accuracy,
    'Precision': precision,
    'Recall': recall,
    'F1 Score': f1
})

results_df = pd.DataFrame(results)

display(results_df)

```

	Feature Representation	Model	Accuracy	Precision	Recall	F1 Score
0	Bag of Words	Logistic Regression	0.859504	0.886832	0.859504	0.852497

```

In [15]: # =====
# Cell 12 - TF-IDF Representation
# =====

from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(
    ngram_range=(1, 2)
)

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

print("TF-IDF created successfully.")
print("Training matrix shape:", X_train_tfidf.shape)
print("Testing matrix shape:", X_test_tfidf.shape)
print("Vocabulary size:", len(tfidf_vectorizer.vocabulary_))

```

TF-IDF created successfully.
 Training matrix shape: (484, 6139)
 Testing matrix shape: (121, 6139)
 Vocabulary size: 6139

```

In [16]: # =====
# Cell 13 - Random Oversampling
# TF-IDF Training Data
# =====

X_train_tfidf_balanced, y_train_tfidf_balanced = ros.fit_resample(
    X_train_tfidf,
    y_train
)

print("Before Oversampling:")
print(Counter(y_train))

```

```
print("\nAfter Oversampling:")
print(Counter(y_train_tfidf_balanced))

print("\nBalanced TF-IDF training shape:")
print(X_train_tfidf_balanced.shape)
```

Before Oversampling:

```
Counter({0: 220, 1: 184, 2: 80})
```

After Oversampling:

```
Counter({0: 220, 2: 220, 1: 220})
```

Balanced TF-IDF training shape:

```
(660, 6139)
```

```
In [17]: # =====
# Cell 14 - Logistic Regression
# TF-IDF without Oversampling
# =====

lr_tfidf = LogisticRegression(
    max_iter=1000,
    random_state=42
)

# Train on original TF-IDF training data
lr_tfidf.fit(X_train_tfidf, y_train)

# Predict
y_pred_lr_tfidf = lr_tfidf.predict(X_test_tfidf)

# Evaluation
accuracy_tfidf = accuracy_score(y_test, y_pred_lr_tfidf)

precision_tfidf = precision_score(
    y_test,
    y_pred_lr_tfidf,
    average='weighted',
    zero_division=0
)

recall_tfidf = recall_score(
    y_test,
    y_pred_lr_tfidf,
    average='weighted',
    zero_division=0
)

f1_tfidf = f1_score(
    y_test,
    y_pred_lr_tfidf,
    average='weighted',
    zero_division=0
)
```

```

print("Logistic Regression - TF-IDF")
print("-----")
print(f"Accuracy : {accuracy_tfidf:.4f}")
print(f"Precision: {precision_tfidf:.4f}")
print(f"Recall   : {recall_tfidf:.4f}")
print(f"F1 Score : {f1_tfidf:.4f}")

print("\nClassification Report:")
print(
    classification_report(
        y_test,
        y_pred_lr_tfidf,
        target_names=[
            "Labor Sector",
            "Civil Service Sector",
            "Social Development Sector"
        ],
        zero_division=0
    )
)

cm_tfidf = confusion_matrix(y_test, y_pred_lr_tfidf)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm_tfidf,
    display_labels=[
        "Labor",
        "Civil Service",
        "Social Development"
    ]
)

disp.plot()
plt.title("Logistic Regression - TF-IDF")
plt.show()

```

Logistic Regression - TF-IDF

Accuracy : 0.8017

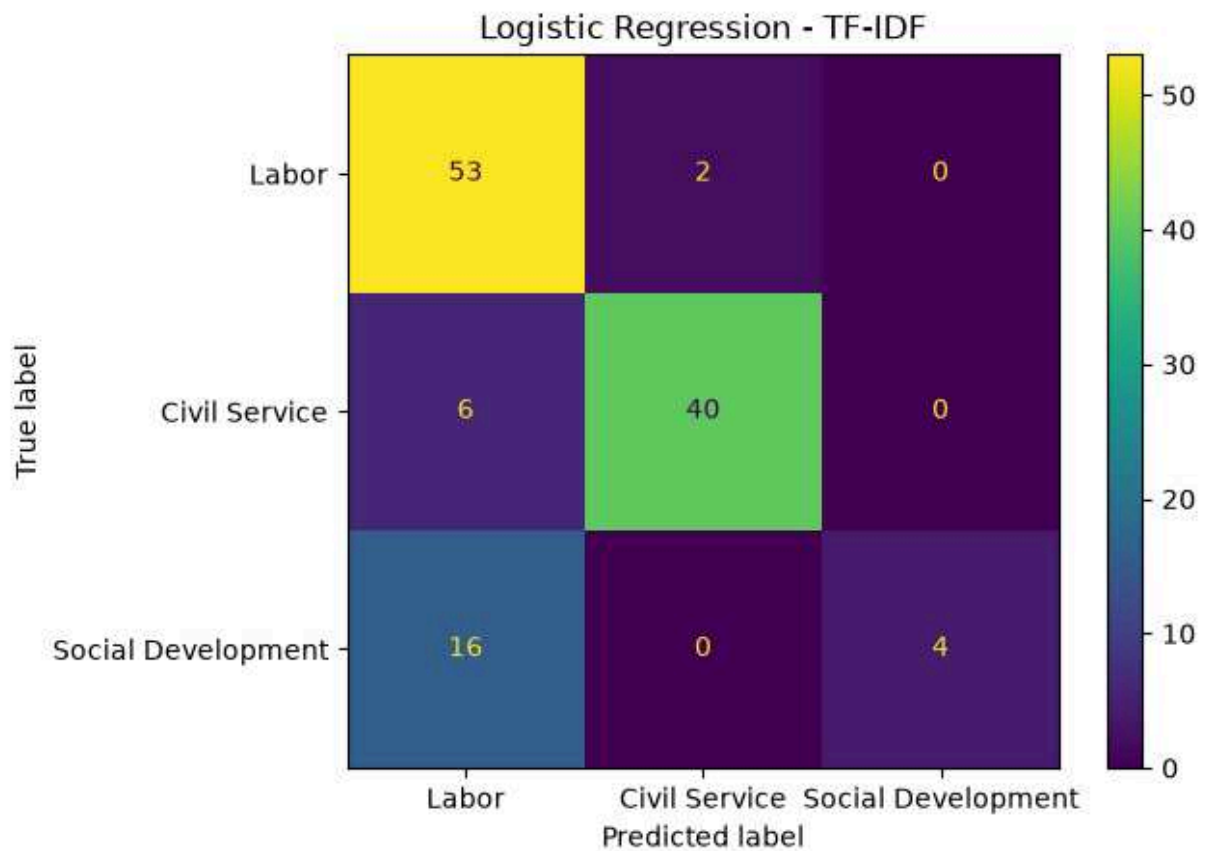
Precision: 0.8486

Recall : 0.8017

F1 Score : 0.7713

Classification Report:

	precision	recall	f1-score	support
Labor Sector	0.71	0.96	0.82	55
Civil Service Sector	0.95	0.87	0.91	46
Social Development Sector	1.00	0.20	0.33	20
accuracy			0.80	121
macro avg	0.89	0.68	0.69	121
weighted avg	0.85	0.80	0.77	121



```
In [18]: # =====
# Cell 15 - Logistic Regression
# TF-IDF + Random Oversampling
# =====

lr_tfidf_ros = LogisticRegression(
    max_iter=1000,
    random_state=42
)

# Train on balanced TF-IDF data
lr_tfidf_ros.fit(
    X_train_tfidf_balanced,
    y_train_tfidf_balanced
)

# Predict on original test set
y_pred_lr_tfidf_ros = lr_tfidf_ros.predict(X_test_tfidf)

# Evaluation
accuracy_tfidf_ros = accuracy_score(
    y_test,
    y_pred_lr_tfidf_ros
)

precision_tfidf_ros = precision_score(
    y_test,
    y_pred_lr_tfidf_ros,
    average='weighted',
```

```

        zero_division=0
    )

recall_tfidf_ros = recall_score(
    y_test,
    y_pred_lr_tfidf_ros,
    average='weighted',
    zero_division=0
)

f1_tfidf_ros = f1_score(
    y_test,
    y_pred_lr_tfidf_ros,
    average='weighted',
    zero_division=0
)

print("Logistic Regression - TF-IDF + Random Oversampling")
print("-----")
print(f"Accuracy : {accuracy_tfidf_ros:.4f}")
print(f"Precision: {precision_tfidf_ros:.4f}")
print(f"Recall   : {recall_tfidf_ros:.4f}")
print(f"F1 Score : {f1_tfidf_ros:.4f}")

print("\nClassification Report:")
print(
    classification_report(
        y_test,
        y_pred_lr_tfidf_ros,
        target_names=[
            "Labor Sector",
            "Civil Service Sector",
            "Social Development Sector"
        ],
        zero_division=0
    )
)

cm_tfidf_ros = confusion_matrix(
    y_test,
    y_pred_lr_tfidf_ros
)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm_tfidf_ros,
    display_labels=[
        "Labor",
        "Civil Service",
        "Social Development"
    ]
)

disp.plot()
plt.title("Logistic Regression - TF-IDF + Oversampling")
plt.show()

```

Logistic Regression - TF-IDF + Random Oversampling

Accuracy : 0.8678

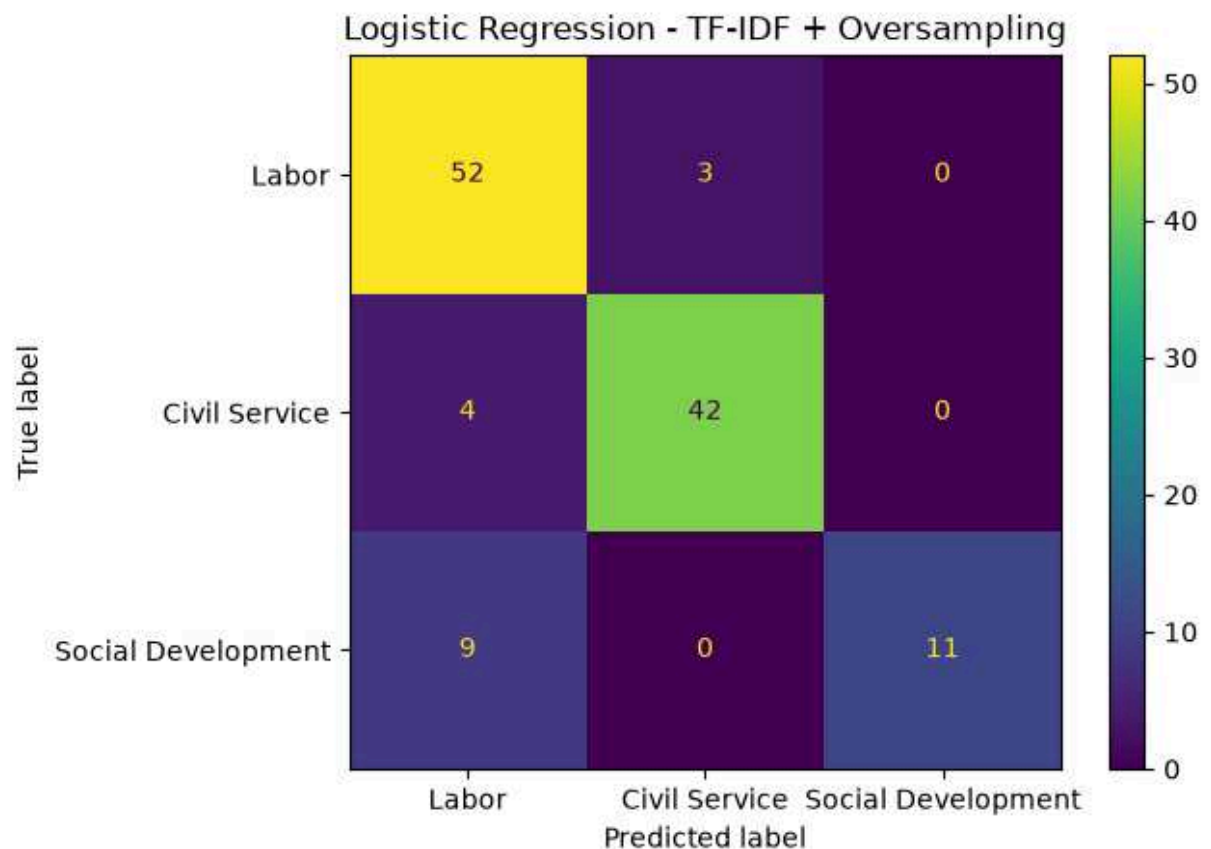
Precision: 0.8837

Recall : 0.8678

F1 Score : 0.8622

Classification Report:

	precision	recall	f1-score	support
Labor Sector	0.80	0.95	0.87	55
Civil Service Sector	0.93	0.91	0.92	46
Social Development Sector	1.00	0.55	0.71	20
accuracy			0.87	121
macro avg	0.91	0.80	0.83	121
weighted avg	0.88	0.87	0.86	121



```
In [19]: # =====
# Cell 16 - Support Vector Machine
# TF-IDF + Random Oversampling
# =====

from sklearn.svm import SVC

svm_tfidf_ros = SVC(
    kernel='linear',
    random_state=42
)
```

```

# Train on balanced TF-IDF data
svm_tfidf_ros.fit(
    X_train_tfidf_balanced,
    y_train_tfidf_balanced
)

# Predict on original test data
y_pred_svm_tfidf_ros = svm_tfidf_ros.predict(X_test_tfidf)

# Evaluation
accuracy_svm_tfidf_ros = accuracy_score(
    y_test,
    y_pred_svm_tfidf_ros
)

precision_svm_tfidf_ros = precision_score(
    y_test,
    y_pred_svm_tfidf_ros,
    average='weighted',
    zero_division=0
)

recall_svm_tfidf_ros = recall_score(
    y_test,
    y_pred_svm_tfidf_ros,
    average='weighted',
    zero_division=0
)

f1_svm_tfidf_ros = f1_score(
    y_test,
    y_pred_svm_tfidf_ros,
    average='weighted',
    zero_division=0
)

print("SVM - TF-IDF + Random Oversampling")
print("-----")
print(f"Accuracy : {accuracy_svm_tfidf_ros:.4f}")
print(f"Precision: {precision_svm_tfidf_ros:.4f}")
print(f"Recall   : {recall_svm_tfidf_ros:.4f}")
print(f"F1 Score : {f1_svm_tfidf_ros:.4f}")

print("\nClassification Report:")
print(
    classification_report(
        y_test,
        y_pred_svm_tfidf_ros,
        target_names=[
            "Labor Sector",
            "Civil Service Sector",
            "Social Development Sector"
        ],
        zero_division=0
    )
)

```

```

)

# Confusion Matrix
cm_svm_tfidf_ros = confusion_matrix(
    y_test,
    y_pred_svm_tfidf_ros
)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm_svm_tfidf_ros,
    display_labels=[
        "Labor",
        "Civil Service",
        "Social Development"
    ]
)

disp.plot()
plt.title("SVM - TF-IDF + Oversampling")
plt.show()

```

SVM - TF-IDF + Random Oversampling

Accuracy : 0.9091

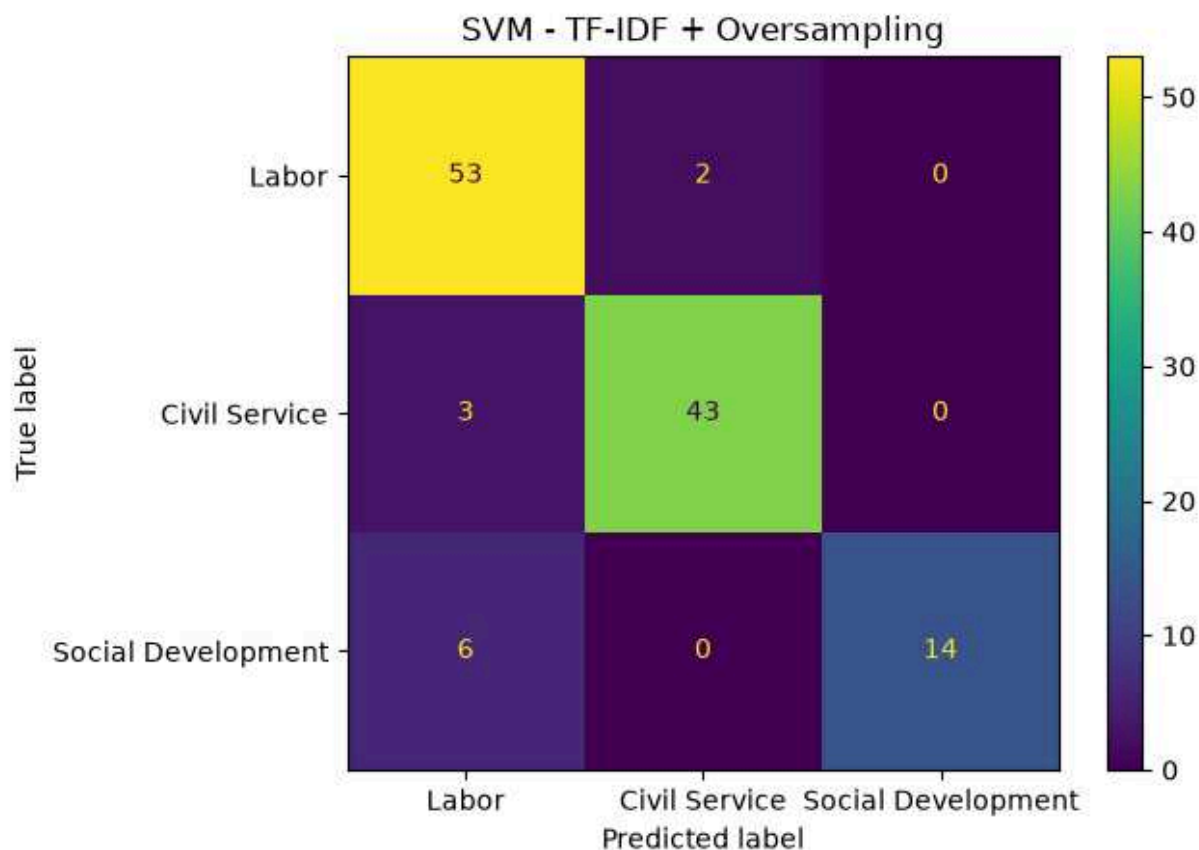
Precision: 0.9171

Recall : 0.9091

F1 Score : 0.9072

Classification Report:

	precision	recall	f1-score	support
Labor Sector	0.85	0.96	0.91	55
Civil Service Sector	0.96	0.93	0.95	46
Social Development Sector	1.00	0.70	0.82	20
accuracy			0.91	121
macro avg	0.94	0.87	0.89	121
weighted avg	0.92	0.91	0.91	121



```
In [20]: # =====
# Cell 17 - Random Forest
# TF-IDF + Random Oversampling
# =====

from sklearn.ensemble import RandomForestClassifier

rf_tfidf_ros = RandomForestClassifier(
    n_estimators=200,
    random_state=42
)

# Train on balanced TF-IDF data
rf_tfidf_ros.fit(
    X_train_tfidf_balanced,
    y_train_tfidf_balanced
)

# Predict on original test data
y_pred_rf_tfidf_ros = rf_tfidf_ros.predict(X_test_tfidf)

# Evaluation
accuracy_rf_tfidf_ros = accuracy_score(
    y_test,
    y_pred_rf_tfidf_ros
)

precision_rf_tfidf_ros = precision_score(
    y_test,
```

```

    y_pred_rf_tfidf_ros,
    average='weighted',
    zero_division=0
)

recall_rf_tfidf_ros = recall_score(
    y_test,
    y_pred_rf_tfidf_ros,
    average='weighted',
    zero_division=0
)

f1_rf_tfidf_ros = f1_score(
    y_test,
    y_pred_rf_tfidf_ros,
    average='weighted',
    zero_division=0
)

print("Random Forest - TF-IDF + Random Oversampling")
print("-----")
print(f"Accuracy : {accuracy_rf_tfidf_ros:.4f}")
print(f"Precision: {precision_rf_tfidf_ros:.4f}")
print(f"Recall   : {recall_rf_tfidf_ros:.4f}")
print(f"F1 Score : {f1_rf_tfidf_ros:.4f}")

print("\nClassification Report:")
print(
    classification_report(
        y_test,
        y_pred_rf_tfidf_ros,
        target_names=[
            "Labor Sector",
            "Civil Service Sector",
            "Social Development Sector"
        ],
        zero_division=0
    )
)

# Confusion Matrix
cm_rf_tfidf_ros = confusion_matrix(
    y_test,
    y_pred_rf_tfidf_ros
)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm_rf_tfidf_ros,
    display_labels=[
        "Labor",
        "Civil Service",
        "Social Development"
    ]
)

disp.plot()

```

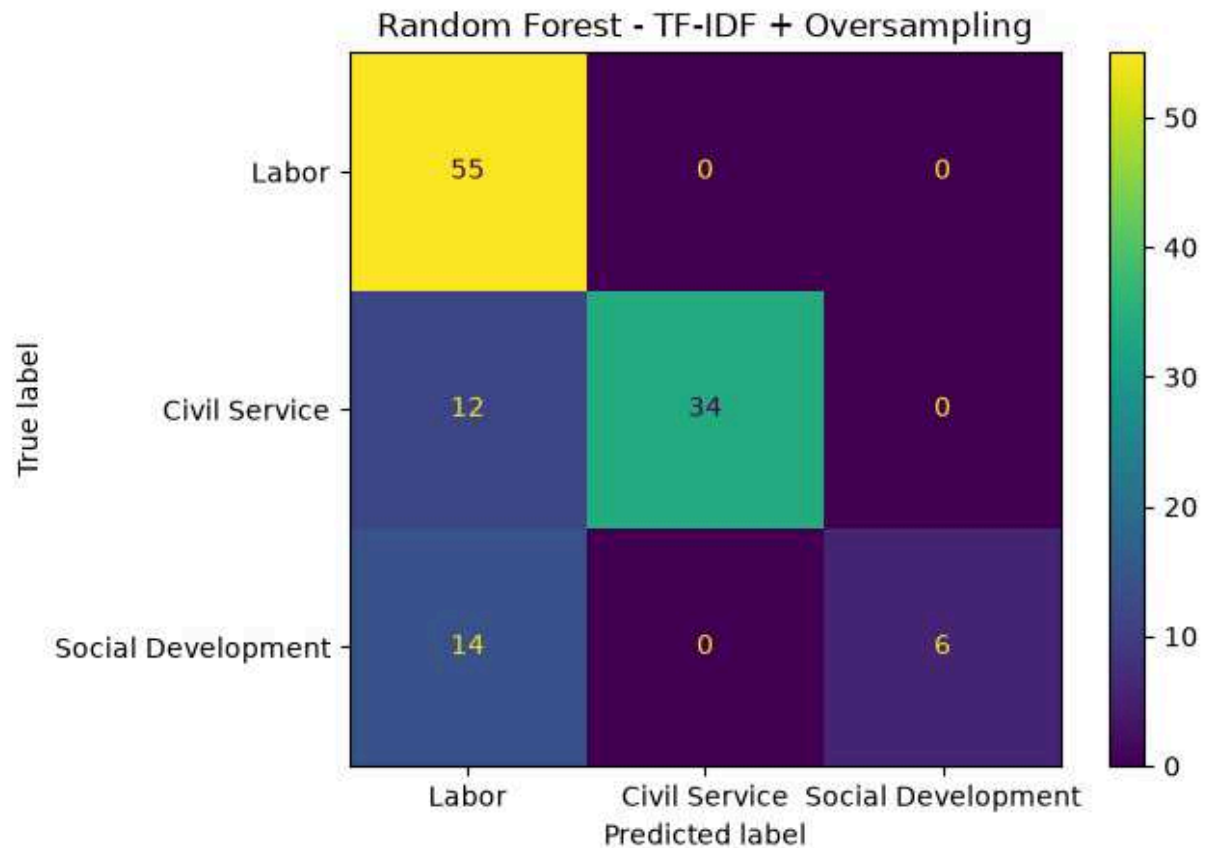
```
plt.title("Random Forest - TF-IDF + Oversampling")
plt.show()
```

Random Forest - TF-IDF + Random Oversampling

 Accuracy : 0.7851
 Precision: 0.8541
 Recall : 0.7851
 F1 Score : 0.7671

Classification Report:

	precision	recall	f1-score	support
Labor Sector	0.68	1.00	0.81	55
Civil Service Sector	1.00	0.74	0.85	46
Social Development Sector	1.00	0.30	0.46	20
accuracy			0.79	121
macro avg	0.89	0.68	0.71	121
weighted avg	0.85	0.79	0.77	121



```
In [21]: # =====
# Cell 18 - Decision Tree
# TF-IDF + Random Oversampling
# =====

from sklearn.tree import DecisionTreeClassifier

dt_tfidf_ros = DecisionTreeClassifier(
```

```

    random_state=42
)

# Train on balanced TF-IDF data
dt_tfidf_ros.fit(
    X_train_tfidf_balanced,
    y_train_tfidf_balanced
)

# Predict on original test data
y_pred_dt_tfidf_ros = dt_tfidf_ros.predict(X_test_tfidf)

# Evaluation
accuracy_dt_tfidf_ros = accuracy_score(
    y_test,
    y_pred_dt_tfidf_ros
)

precision_dt_tfidf_ros = precision_score(
    y_test,
    y_pred_dt_tfidf_ros,
    average='weighted',
    zero_division=0
)

recall_dt_tfidf_ros = recall_score(
    y_test,
    y_pred_dt_tfidf_ros,
    average='weighted',
    zero_division=0
)

f1_dt_tfidf_ros = f1_score(
    y_test,
    y_pred_dt_tfidf_ros,
    average='weighted',
    zero_division=0
)

print("Decision Tree - TF-IDF + Random Oversampling")
print("-----")
print(f"Accuracy : {accuracy_dt_tfidf_ros:.4f}")
print(f"Precision: {precision_dt_tfidf_ros:.4f}")
print(f"Recall    : {recall_dt_tfidf_ros:.4f}")
print(f"F1 Score  : {f1_dt_tfidf_ros:.4f}")

print("\nClassification Report:")
print(
    classification_report(
        y_test,
        y_pred_dt_tfidf_ros,
        target_names=[
            "Labor Sector",
            "Civil Service Sector",
            "Social Development Sector"
        ],
    )

```

```

        zero_division=0
    )
)

# Confusion Matrix
cm_dt_tfidf_ros = confusion_matrix(
    y_test,
    y_pred_dt_tfidf_ros
)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm_dt_tfidf_ros,
    display_labels=[
        "Labor",
        "Civil Service",
        "Social Development"
    ]
)

disp.plot()
plt.title("Decision Tree - TF-IDF + Oversampling")
plt.show()

```

Decision Tree - TF-IDF + Random Oversampling

Accuracy : 0.7355

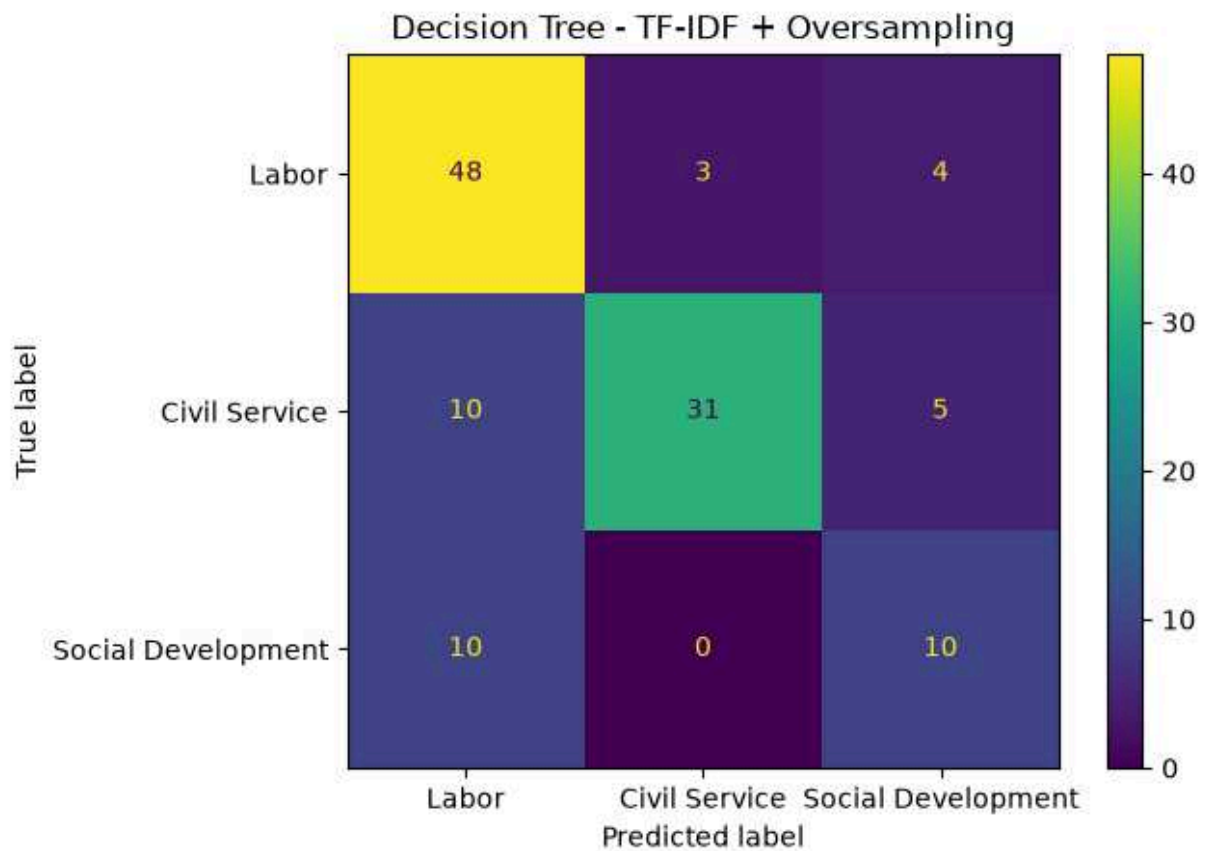
Precision: 0.7545

Recall : 0.7355

F1 Score : 0.7342

Classification Report:

	precision	recall	f1-score	support
Labor Sector	0.71	0.87	0.78	55
Civil Service Sector	0.91	0.67	0.78	46
Social Development Sector	0.53	0.50	0.51	20
accuracy			0.74	121
macro avg	0.71	0.68	0.69	121
weighted avg	0.75	0.74	0.73	121



```
In [22]: # =====
# Cell 20 - SVM
# TF-IDF without Oversampling
# =====

svm_tfidf = SVC(
    kernel='linear',
    random_state=42
)

# Train on original TF-IDF training data
svm_tfidf.fit(
    X_train_tfidf,
    y_train
)

# Predict on original test data
y_pred_svm_tfidf = svm_tfidf.predict(X_test_tfidf)

# Evaluation
accuracy_svm_tfidf = accuracy_score(
    y_test,
    y_pred_svm_tfidf
)

precision_svm_tfidf = precision_score(
    y_test,
    y_pred_svm_tfidf,
    average='weighted',
```

```

        zero_division=0
    )

recall_svm_tfidf = recall_score(
    y_test,
    y_pred_svm_tfidf,
    average='weighted',
    zero_division=0
)

f1_svm_tfidf = f1_score(
    y_test,
    y_pred_svm_tfidf,
    average='weighted',
    zero_division=0
)

print("SVM - TF-IDF without Oversampling")
print("-----")
print(f"Accuracy : {accuracy_svm_tfidf:.4f}")
print(f"Precision: {precision_svm_tfidf:.4f}")
print(f"Recall   : {recall_svm_tfidf:.4f}")
print(f"F1 Score : {f1_svm_tfidf:.4f}")

print("\nClassification Report:")
print(
    classification_report(
        y_test,
        y_pred_svm_tfidf,
        target_names=[
            "Labor Sector",
            "Civil Service Sector",
            "Social Development Sector"
        ],
        zero_division=0
    )
)

cm_svm_tfidf = confusion_matrix(
    y_test,
    y_pred_svm_tfidf
)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm_svm_tfidf,
    display_labels=[
        "Labor",
        "Civil Service",
        "Social Development"
    ]
)

disp.plot()
plt.title("SVM - TF-IDF without Oversampling")
plt.show()

```

SVM - TF-IDF without Oversampling

Accuracy : 0.9008

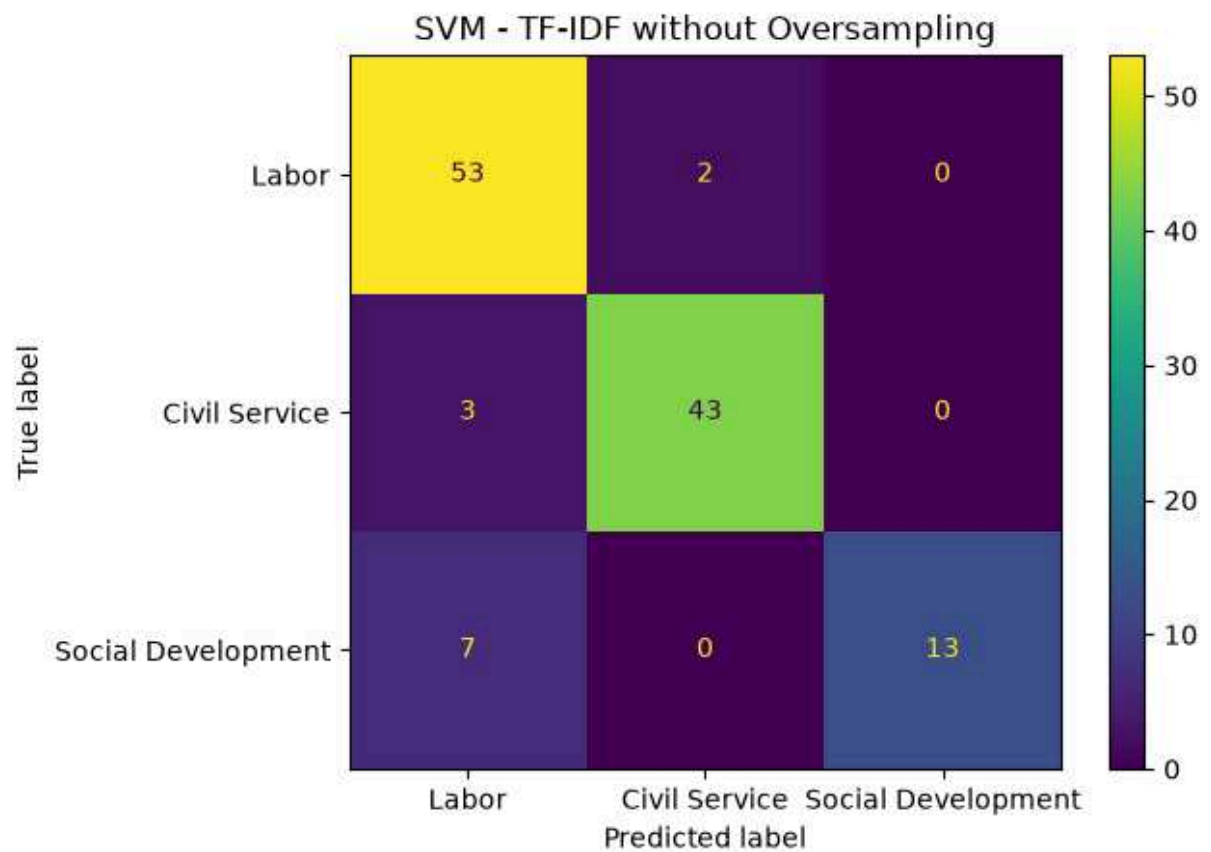
Precision: 0.9110

Recall : 0.9008

F1 Score : 0.8978

Classification Report:

	precision	recall	f1-score	support
Labor Sector	0.84	0.96	0.90	55
Civil Service Sector	0.96	0.93	0.95	46
Social Development Sector	1.00	0.65	0.79	20
accuracy			0.90	121
macro avg	0.93	0.85	0.88	121
weighted avg	0.91	0.90	0.90	121



```
In [23]: # =====
# Cell 21 - Updated Final Comparison
# =====

comparison_results = pd.DataFrame([
    {
        "Feature Representation": "Bag of Words",
        "Balancing": "No",
        "Model": "Logistic Regression",
        "Accuracy": accuracy,
        "Precision": precision,
```

```

    "Recall": recall,
    "F1 Score": f1
  },
  {
    "Feature Representation": "Bag of Words",
    "Balancing": "Random Oversampling",
    "Model": "Logistic Regression",
    "Accuracy": accuracy_ros,
    "Precision": precision_ros,
    "Recall": recall_ros,
    "F1 Score": f1_ros
  },
  {
    "Feature Representation": "TF-IDF (1,2)",
    "Balancing": "No",
    "Model": "Logistic Regression",
    "Accuracy": accuracy_tfidf,
    "Precision": precision_tfidf,
    "Recall": recall_tfidf,
    "F1 Score": f1_tfidf
  },
  {
    "Feature Representation": "TF-IDF (1,2)",
    "Balancing": "Random Oversampling",
    "Model": "Logistic Regression",
    "Accuracy": accuracy_tfidf_ros,
    "Precision": precision_tfidf_ros,
    "Recall": recall_tfidf_ros,
    "F1 Score": f1_tfidf_ros
  },
  {
    "Feature Representation": "TF-IDF (1,2)",
    "Balancing": "No",
    "Model": "SVM",
    "Accuracy": accuracy_svm_tfidf,
    "Precision": precision_svm_tfidf,
    "Recall": recall_svm_tfidf,
    "F1 Score": f1_svm_tfidf
  },
  {
    "Feature Representation": "TF-IDF (1,2)",
    "Balancing": "Random Oversampling",
    "Model": "SVM",
    "Accuracy": accuracy_svm_tfidf_ros,
    "Precision": precision_svm_tfidf_ros,
    "Recall": recall_svm_tfidf_ros,
    "F1 Score": f1_svm_tfidf_ros
  },
  {
    "Feature Representation": "TF-IDF (1,2)",
    "Balancing": "Random Oversampling",
    "Model": "Random Forest",
    "Accuracy": accuracy_rf_tfidf_ros,
    "Precision": precision_rf_tfidf_ros,
    "Recall": recall_rf_tfidf_ros,
    "F1 Score": f1_rf_tfidf_ros
  }

```

```

    },
    {
        "Feature Representation": "TF-IDF (1,2)",
        "Balancing": "Random Oversampling",
        "Model": "Decision Tree",
        "Accuracy": accuracy_dt_tfidf_ros,
        "Precision": precision_dt_tfidf_ros,
        "Recall": recall_dt_tfidf_ros,
        "F1 Score": f1_dt_tfidf_ros
    }
])

comparison_results = comparison_results.sort_values(
    by="F1 Score",
    ascending=False
).reset_index(drop=True)

display(
    comparison_results.style.format({
        "Accuracy": "{:.4f}",
        "Precision": "{:.4f}",
        "Recall": "{:.4f}",
        "F1 Score": "{:.4f}"
    })
)

best_model = comparison_results.iloc[0]

print("\n=====")
print("BEST MODEL")
print("=====")
print("Model:", best_model["Model"])
print("Feature Representation:", best_model["Feature Representation"])
print("Balancing:", best_model["Balancing"])
print(f"Accuracy: {best_model['Accuracy']:.4f}")
print(f"Precision: {best_model['Precision']:.4f}")
print(f"Recall: {best_model['Recall']:.4f}")
print(f"F1 Score: {best_model['F1 Score']:.4f}")

```

	Feature Representation	Balancing	Model	Accuracy	Precision	Recall	F1 Score
0	TF-IDF (1,2)	Random Oversampling	SVM	0.9091	0.9171	0.9091	0.9072
1	TF-IDF (1,2)	No	SVM	0.9008	0.9110	0.9008	0.8978
2	TF-IDF (1,2)	Random Oversampling	Logistic Regression	0.8678	0.8837	0.8678	0.8622
3	Bag of Words	No	Logistic Regression	0.8595	0.8868	0.8595	0.8525
4	Bag of Words	Random Oversampling	Logistic Regression	0.8512	0.8601	0.8512	0.8467
5	TF-IDF (1,2)	No	Logistic Regression	0.8017	0.8486	0.8017	0.7713
6	TF-IDF (1,2)	Random Oversampling	Random Forest	0.7851	0.8541	0.7851	0.7671
7	TF-IDF (1,2)	Random Oversampling	Decision Tree	0.7355	0.7545	0.7355	0.7342

=====

BEST MODEL

=====

Model: SVM

Feature Representation: TF-IDF (1,2)

Balancing: Random Oversampling

Accuracy: 0.9091

Precision: 0.9171

Recall: 0.9091

F1 Score: 0.9072

```
In [24]: # =====
# Final Model Comparison Plot
# =====

import matplotlib.pyplot as plt

plot_df = comparison_results.copy()

# Create a readable experiment name
plot_df["Experiment"] = (
    plot_df["Model"]
    + " | "
    + plot_df["Feature Representation"]
    + " | "
    + plot_df["Balancing"]
)

# Sort by F1 Score from highest to lowest
plot_df = plot_df.sort_values(
    by="F1 Score",
    ascending=False
```

```

).reset_index(drop=True)

# Create horizontal bar chart
plt.figure(figsize=(11, 7))

bars = plt.barh(
    plot_df["Experiment"],
    plot_df["F1 Score"]
)

# Put the best model at the top
plt.gca().invert_yaxis()

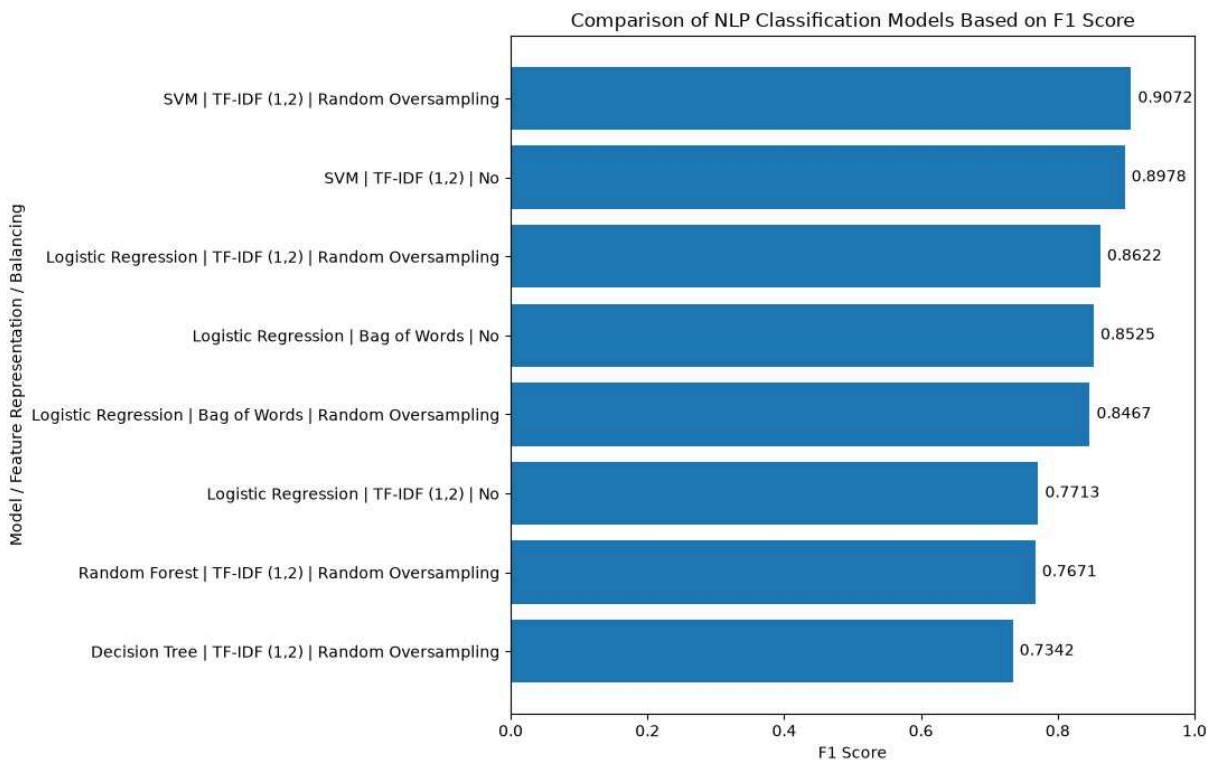
# Add numeric F1 values
for bar, score in zip(bars, plot_df["F1 Score"]):
    plt.text(
        score + 0.01,
        bar.get_y() + bar.get_height() / 2,
        f"{score:.4f}",
        va="center"
    )

# Labels and title
plt.xlabel("F1 Score")
plt.ylabel("Model / Feature Representation / Balancing")
plt.title("Comparison of NLP Classification Models Based on F1 Score")

# Keep the metric scale consistent
plt.xlim(0, 1.0)

plt.tight_layout()
plt.show()

```



Conclusion

In this assignment, an Arabic NLP classification system was developed to classify HRSD questions into three sectors: Labor Sector, Civil Service Sector, and Social Development Sector.

The dataset was cleaned by removing missing and duplicated questions, followed by Arabic text preprocessing and an 80/20 stratified train-test split.

Two feature representation approaches were evaluated: Bag of Words and TF-IDF using unigrams and bigrams. Random Oversampling was also applied to reduce the effect of class imbalance.

Several machine learning models were evaluated, including Logistic Regression, Support Vector Machine, Random Forest, and Decision Tree. The models were compared using Accuracy, Precision, Recall, F1-score, and Confusion Matrix.

The best-performing model was Support Vector Machine (SVM) with TF-IDF and Random Oversampling, achieving:

Accuracy: 90.91% Precision: 91.71% Recall: 90.91% F1-score: 90.72%

Random Oversampling also improved the recall of the minority Social Development class from 65% to 70% when using SVM, demonstrating the benefit of handling class imbalance.

Overall, the results indicate that TF-IDF combined with SVM provides the most effective approach for classifying Arabic HRSD questions in this dataset