

Arabic Sentiment Analysis using Machine Learning Models

Lecture 08

Prepared by

Dr. Hager Saleh

Dataset linking

Arabic Sentiment Analysis:

<https://www.kaggle.com/code/mohamedabdelmohsen/preprocessing-techniques-on-arabic-text-modeling/input>

Import necessary libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
```

Function to evaluate model

```
def evaluate_model(model_name, y_true, y_pred):

    # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    # Create a report
    report = classification_report(y_true, y_pred)

    # Output results
    metrics = {
        'Model Name': model_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,

        'Classification Report': report
    }

    # Plot Confusion Matrix
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=np.unique(y_true),
                yticklabels=np.unique(y_true))
    plt.title(f'Confusion Matrix for {model_name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

    return metrics
```

Reading dataset

```
df = pd.read_csv("Arabic Sentiment Analysis Dataset - SS2030.csv")
df.head()
```

	text	Sentiment
0	حقوق المرأة https://t.co/Mzf90Ta5g1	1
1	RT @___IHAVENOIDEA: حقوق المرأة في الإسلام. ht...	1

```
2 RT @saud_talep: Retweeted 1 لجنة التنمية بشيرا (...)  
3 RT @MojKsa: 1 حقوق المرأة التي تضمنها لها وزارة ...  
4 RT @abm112211: 1 ولي امر الزوجة او ولي الزوجة او...
```

Preprocessing arabic text

```
import re
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

def preprocess_arabic_text(text):
    # 1. Remove diacritics (tashkeel)
    diacritics = re.compile(r'[\u0617-\u061A\u064B-\u0652]')
    text = re.sub(diacritics, '', text)

    # 2. Normalize Arabic letters
    text = re.sub(r'[\u0621\u0622\u0623\u0624\u0625\u0626\u0627\u0628\u0629\u0630\u0631\u0632\u0633\u0634\u0635\u0636\u0637\u0638\u0639\u0640\u0641\u0642\u0643\u0644\u0645\u0646\u0647\u0648\u0649\u0650\u0651\u0652\u0653\u0654\u0655\u0656\u0657\u0658\u0659\u0660\u0661\u0662\u0663\u0664\u0665\u0666\u0667\u0668\u0669\u0670\u0671\u0672\u0673\u0674\u0675\u0676\u0677\u0678\u0679\u0680\u0681\u0682\u0683\u0684\u0685\u0686\u0687\u0688\u0689\u0690\u0691\u0692\u0693\u0694\u0695\u0696\u0697\u0698\u0699\u06A0\u06A1\u06A2\u06A3\u06A4\u06A5\u06A6\u06A7\u06A8\u06A9\u06B0\u06B1\u06B2\u06B3\u06B4\u06B5\u06B6\u06B7\u06B8\u06B9\u06C0\u06C1\u06C2\u06C3\u06C4\u06C5\u06C6\u06C7\u06C8\u06C9\u06D0\u06D1\u06D2\u06D3\u06D4\u06D5\u06D6\u06D7\u06D8\u06D9\u06E0\u06E1\u06E2\u06E3\u06E4\u06E5\u06E6\u06E7\u06E8\u06E9\u06F0\u06F1\u06F2\u06F3\u06F4\u06F5\u06F6\u06F7\u06F8\u06F9\u06FA\u06FB\u06FC\u06FD\u06FE\u06FF]', 'l', text)
    text = re.sub(r'ي', 'ى', text)
    text = re.sub(r'و', 'ؤ', text)
    text = re.sub(r'ي', 'ئ', text)
    text = re.sub(r'ه', 'ة', text)
    text = re.sub(r'ك', 'گ', text)

    # 3. Remove Tatweel (elongation)
    text = re.sub(r'_{1,}', '', text)

    # 4. Remove non-Arabic characters and punctuation
    text = re.sub(r'[A-Za-z0-9]', '', text)
    text = re.sub(r'^[\u0600-\u06FF\s]', '', text)

    # 5. Tokenize
    tokens = word_tokenize(text)

    # 6. Remove stopwords
    arabic_stopwords = set(stopwords.words("arabic"))
    tokens = [word for word in tokens if word not in arabic_stopwords]

    clean_text = ' '.join(tokens)
    return clean_text
```

```
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\compumarts\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\compumarts\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

```

df["clean_text"] = df["text"].apply(preprocess_arabic_text)
df = df.dropna()

df.head()

```

	text	Sentiment	\
0	حقوق المرأة https://t.co/Mzf90Ta5g1	1	
1	RT @__IHAVENOIDEA: حقوق المرأة في الإسلام. ht...	1	
2	RT @saud_talep: Retweeted 1 ... لجنة التنمية بشبرا		
3	RT @MojKsa: 1 ... حقوق المرأة التي تضمنها لها وزارة		
4	RT @abm112211: 1 ... ولي امر الزوجة او ولي الزوجة او...		

```

df["clean_text"]

```

	clean_text
0	حقوق المراة
1	حقوق المراة الاسلام
2	... لجنة التنمية بشبرا زال التسجيل مستمر دوره حقوق
3	حقوق المراة تضمنها وزاره العدل
4	... ولي امر الزوجة او ولي الزوجة او ولي المراة الا

```

df['Sentiment'].value_counts()

Sentiment
1    2436
0    1816
Name: count, dtype: int64

X = df['clean_text']
y = df['Sentiment']

```

Split data into training and testing sets

```

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

```

Feature representation

Bag word

```

vectorizer1 = CountVectorizer()
X_train_Bag_word = vectorizer1.fit_transform(X_train)
X_test_Bag_word = vectorizer1.transform(X_test)

```

Random Forest Classifier

```

# Define the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Define the hyperparameters for grid search

```

```

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier,
    param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train_Bag_word, y_train)

# Get the best estimator from grid search
best_rf_classifier = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_rf_classifier.predict(X_test_Bag_word)

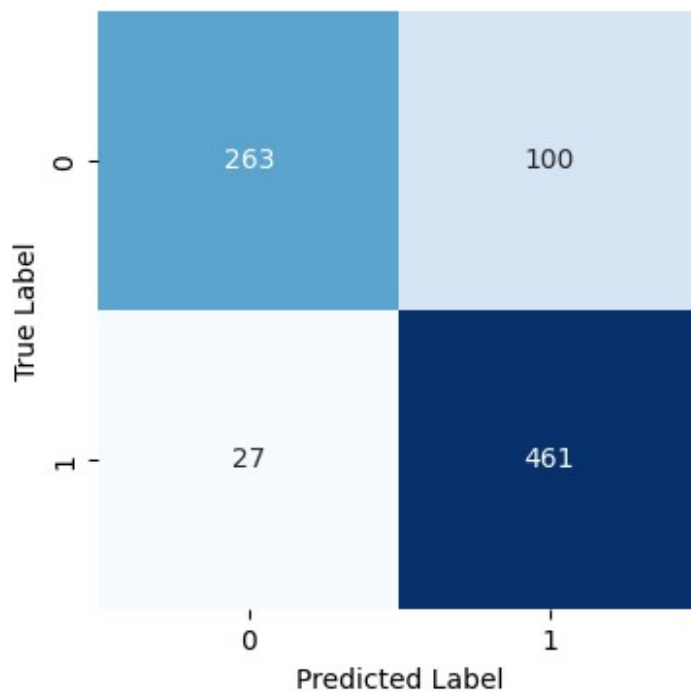
# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test,
    y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
            f"{key}: \n{value}")

# Print the best parameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Confusion Matrix for RandomForestClassifier



Model Name:
RandomForestClassifier
Accuracy: 0.8508
Precision: 0.8581
Recall: 0.8508
F1 Score: 0.8476

	precision	recall	f1-score	support
0	0.91	0.72	0.81	363
1	0.82	0.94	0.88	488
accuracy			0.85	851
macro avg	0.86	0.83	0.84	851
weighted avg	0.86	0.85	0.85	851

Best hyperparameters found by GridSearchCV:
{'max_depth': None, 'n_estimators': 200}

LogisticRegression

```
# Define the Logistic Regression model
LR = LogisticRegression(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
```

```

        'solver': ['liblinear', 'lbfgs'],          # Solver type
    }

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=LR, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train_Bag_word, y_train)

# Get the best estimator from grid search
best_LR = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_LR.predict(X_test_Bag_word)

# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test,
y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```



Model Name:
LogisticRegression
Accuracy: 0.8637
Precision: 0.8662
Recall: 0.8637
F1 Score: 0.8621

	precision	recall	f1-score	support
0	0.89	0.77	0.83	363
1	0.85	0.93	0.89	488
accuracy			0.86	851
macro avg	0.87	0.85	0.86	851
weighted avg	0.87	0.86	0.86	851

Best hyperparameters found by GridSearchCV:
{'solver': 'liblinear'}

Feature representation using TF-IDF with Unigrams, Bigrams, and Trigrams

Three types of n-grams

- Unigrams: "machine", "learning", "fun"
- Bigrams: "machine learning", "deep learning"

- Trigrams: "machine learning is", "deep learning is"
- ngram_range=(min_n, max_n)
 - Unigrams(1,1)
 - Bigrams(2,2)
 - Unigrams + Bigrams (1,2)
 - Unigrams + Bigrams + Trigrams(1,3)

```
vectorizer2 = TfidfVectorizer(ngram_range=(1, 2)) # (1,2) = unigrams + bigrams
X_train_TFIDF = vectorizer2.fit_transform(X_train)
X_test_TFIDF = vectorizer2.transform(X_test)
```

DecisionTreeClassifier

```
# Define the Decision Tree model
DT = DecisionTreeClassifier(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=DT, param_grid=param_grid, cv=10,
    scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train_TFIDF, y_train)

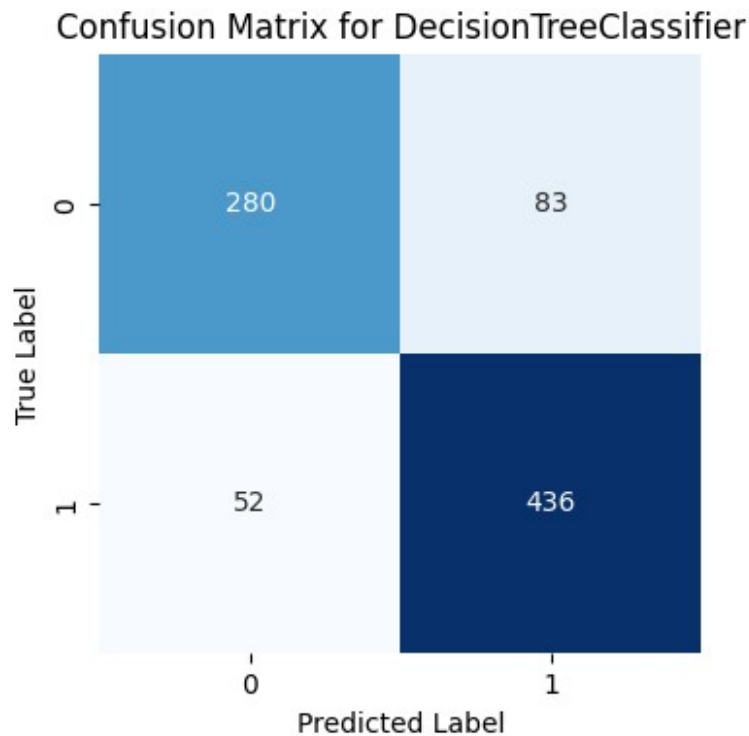
# Get the best estimator from grid search
best_DT = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_DT.predict(X_test_TFIDF)

# Evaluate the model
evaluation_results = evaluate_model('DecisionTreeClassifier', y_test,
    y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
            f"{key}: \n{value}")
```

```
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```



Model Name:
DecisionTreeClassifier
Accuracy: 0.8414
Precision: 0.8415
Recall: 0.8414
F1 Score: 0.8403

	precision	recall	f1-score	support
0	0.84	0.77	0.81	363
1	0.84	0.89	0.87	488
accuracy			0.84	851
macro avg	0.84	0.83	0.84	851
weighted avg	0.84	0.84	0.84	851

Best hyperparameters found by GridSearchCV:
{'criterion': 'entropy', 'max_depth': None}

Support Vector Machine

```

SVM = SVC(random_state=42)

# Define the hyperparameters for grid search
param_grid = {

    'kernel': ['linear', 'rbf', 'poly'],    # Kernel type
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=SVM, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train_TFIDF, y_train)

# Get the best estimator from grid search
best_SVM = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_SVM.predict(X_test_TFIDF)

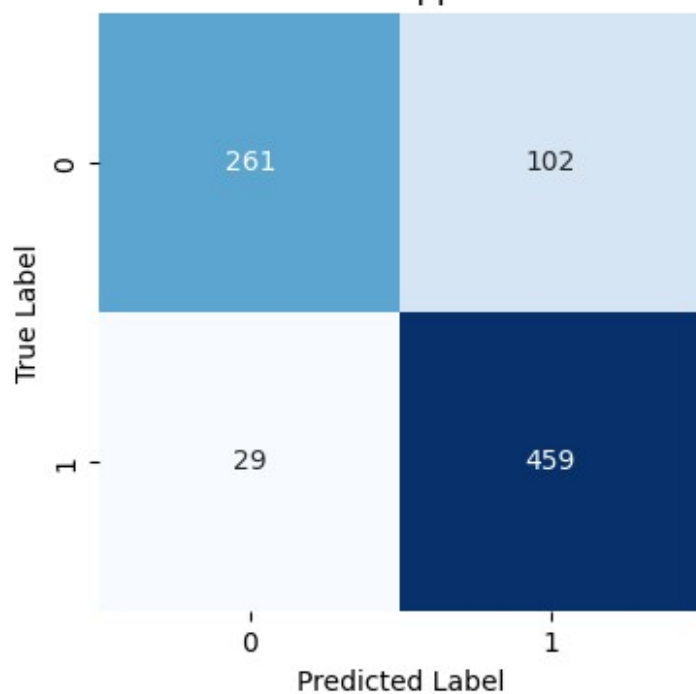
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test,
y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)    # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Confusion Matrix for Support Vector Machine



Model Name:

Support Vector Machine

Accuracy: 0.8461

Precision: 0.8531

Recall: 0.8461

F1 Score: 0.8428

	precision	recall	f1-score	support
0	0.90	0.72	0.80	363
1	0.82	0.94	0.88	488
accuracy			0.85	851
macro avg	0.86	0.83	0.84	851
weighted avg	0.85	0.85	0.84	851

Best hyperparameters found by GridSearchCV:

`{'kernel': 'rbf'}`