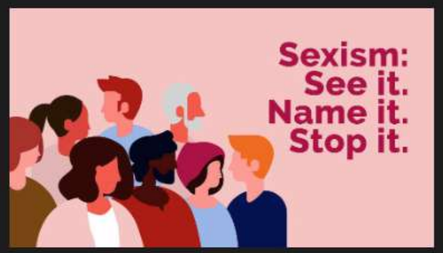
Final Project

Group HOMEWORK. This final project can be collaborative. The maximum members of a group is 2. You can also work by yourself. Please respect the academic integrity. **Remember: if you get caught on cheating, you get F.**

A Introduction to the competition



Sexism is a growing problem online. It can inflict harm on women who are targeted, make online spaces inaccessible and unwelcoming, and perpetuate social asymmetries and injustices. Automated tools are now widely deployed to find, and assess sexist content at scale but most only give classifications for generic, high-level categories, with no further explanation. Flagging what is sexist content and also explaining why it is sexist improves interpretability, trust and understanding of the decisions that automated tools use, empowering both users and moderators.

This project is based on SemEval 2023 - Task 10 - Explainable Detection of Online Sexism (EDOS). Here you can find a detailed introduction to this task.

You only need to complete **TASK A - Binary Sexism Detection: a two-class (or binary) classification where systems have to predict whether a post is sexist or not sexist**. To cut down training time, we only use a subset of the original dataset (5k out of 20k). The dataset can be found in the same folder.

Different from our previous homework, this competition gives you great flexibility (and very few hints), you can determine:

- how to preprocess the input text (e.g., remove emoji, remove stopwords, text lemmatization and stemming, etc.);
- which method to use to encode text features (e.g., TF-IDF, N-grams, Word2vec, GloVe, Part-of-Speech (POS), etc.);
- which model to use.

Requirements

- **Input**: the text for each instance.
- **Output**: the binary label for each instance.
- **Feature engineering**: use at least 2 different methods to extract features and encode text into numerical values.
- **Model selection**: implement with at least 3 different models and compare their performance.
- **Evaluation**: create a dataframe with rows indicating feature+model and columns indicating Precision, Accuracy and F1-score (using weighted average). Your results should have at least 6 rows (2 feature engineering methods x 3 models). Report best performance with (1) your feature engineering method, and (2) the model you choose.
- **Format**: add explainations for each step (you can add markdown cells). At the end of the report, write a summary and answer the following questions:
  - What preprocessing steps do you follow?
  - How do you select the features from the inputs?
  - Which model you use and what is the structure of your model?
  - How do you train your model?
  - What is the performance of your best model?
  - What other models or feature engineering methods would you like to implement in the future?
- **Two Rules**, violations will result in 0 points in the grade:
  - Not allowed to use test set in the training: You CANNOT use any of the instances from test set in the training process.
  - Not allowed to use any generative AI (e.g., ChatGPT).

Evaluation

The performance should be only evaluated on the test set (a total of 1086 instances). Please split original dataset into train set and test set. The test set should NEVER be used in the training process. The evaluation metric is a combination of precision, recall, and f1-score (use `classification_report` in sklearn).

The total points are 10.0. Each team will compete with other teams in the class on their best performance. Points will be deducted if not following the requirements above.

If ALL the requirements are met:

- Top 25% teams: 10.0 points.
- Top 25% - 50% teams: 8.5 points.

The performance should be only evaluated on the test set (a total of 1086 instances). Please split original dataset into train set and test set. The test set should NEVER be used in the training process. The evaluation metric is a combination of precision, recall, and f1-score (use `classification_report` in sklearn).

The total points are 10.0. Each team will compete with other teams in the class on their best performance. Points will be deducted if not following the requirements above.

If ALL the requirements are met:

- Top 25% teams: 10.0 points.
- Top 25% - 50% teams: 8.5 points.
- Top 50% - 75% teams: 7.0 points.
- Top 75% - 100% teams: 6.0 points.

If your best performance is above 0.80 (weighted F1-score) and meets all the requirements, you will also get full points (10.0 points).

⭐ Bonus points will be awarded to top 5 teams (ranked by weighted F1-score):

- Top 1 team: 3pt adding to final grade
- Top 2 team: 2pt adding to final grade
- Top 3-5 teams: 1pt adding to final grade

Submission

Similar as homework, submit both a PDF and .ipynb version of the report.

The report should include:

- (a)code AND outputs
- (b)explainations for each step
- (c)individual experimental results AND combine them in a table
- (d)summary

The due date is **May 2, Thursday by 11:59pm**.

[2]
```python
# Importing necessary libraries
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import LabelEncoder
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk import pos_tag

# Download necessary NLTK resources
nltk.download(['punkt', 'averaged_perceptron_tagger', 'wordnet', 'stopwords'])

# Suppress warnings
import warnings
warnings.filterwarnings("ignore")
```
Pytho

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/sheraliozodov/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/sheraliozodov/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
```

final_project.ipynb

final_project.ipynb > Final Project > Data Preprocessing > # Tokenization and Lemmatization Function

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· py38_env (Python 3.8.1

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/sheraliozodov/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```python
# Load the dataset
data = pd.read_csv('edos_labelled_data.csv')
train_data = data[data['split'] == 'train'].copy()
test_data = data[data['split'] == 'test'].copy()

# Display dataset structure
print("Training Set Shape:", train_data.shape)
print("Testing Set Shape:", test_data.shape)
```

[3] Pytho

```
Training Set Shape: (4193, 4)
Testing Set Shape: (1086, 4)
```

Data Preprocessing

The raw data requires cleaning and preprocessing to be suitable for modeling. We will remove URLs, user tags, non-alphanumeric characters, and numbers from the texts. Additionally, the texts will be tokenized and lemmatized.

```python
# Basic Preprocessing
def preprocess(text):
    text = re.sub(r'http\S+', '', text)  # Remove URLs
    text = re.sub(r'\[USER\]', '', text) # Remove user tags
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = re.sub(r'\s{2,}', ' ', text)  # Replace multiple spaces with a single space
    text = re.sub(r'\d+', '', text)      # Remove numbers
    return text.strip().lower()

def get_wordnet_pos(treebank_tag):
    """Converts treebank tags to wordnet tags."""
    if treebank_tag.startswith('J'):
        return 'a'
    elif treebank_tag.startswith('V'):
        return 'v'
    elif treebank_tag.startswith('N'):
        return 'n'
    elif treebank_tag.startswith('R'):
        return 'r'
    else:
        return 'n'

# Apply preprocessing
train_data['clean_text'] = train_data['text'].apply(preprocess)
test_data['clean_text'] = test_data['text'].apply(preprocess)
```

[4] Pytho

```python
# Tokenization and Lemmatization Function
def lemmatize_text(text):
    """Lemmatizes text using POS tags to ensure accurate lemmatization."""
    words = word_tokenize(text)
    words_pos = pos_tag(words)
    lemmatizer = WordNetLemmatizer()
    lemmatized_words = []
    stop_words = set(stopwords.words('english'))
    for word, pos in words_pos:
        if word not in stop_words:
```

final_project.ipynb

final_project.ipynb > Final Project > Data Preprocessing > # Tokenization and Lemmatization Function

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ···

py38_env (Python 3.8.1

```python
# Tokenization and Lemmatization Function
def lemmatize_text(text):
    """Lemmatizes text using POS tags to ensure accurate lemmatization."""
    words = word_tokenize(text)
    words_pos = pos_tag(words)
    lemmatizer = WordNetLemmatizer()
    lemmatized_words = []
    stop_words = set(stopwords.words('english'))
    for word, pos in words_pos:
        if word not in stop_words:
            lemmatized_word = lemmatizer.lemmatize(word, get_wordnet_pos(pos))
            lemmatized_words.append(lemmatized_word)
    return ' '.join(lemmatized_words)

# Apply tokenization and lemmatization
train_data['lemmatized_text'] = train_data['clean_text'].apply(lemmatize_text)
test_data['lemmatized_text'] = test_data['clean_text'].apply(lemmatize_text)
```

[5] Pytho

Feature Extraction

To represent text data numerically, we utilize two methods: TF-IDF Vectorization and Count Vectorization. Each method transforms the text into a vector that machine learning models can process.

```python
# Feature Extraction
tfidf_vectorizer = TfidfVectorizer(max_features=2000, ngram_range=(1, 2))
X_train_tfidf = tfidf_vectorizer.fit_transform(train_data['lemmatized_text'])
X_test_tfidf = tfidf_vectorizer.transform(test_data['lemmatized_text'])

count_vectorizer = CountVectorizer(max_features=2000, ngram_range=(1, 2))
X_train_count = count_vectorizer.fit_transform(train_data['lemmatized_text'])
X_test_count = count_vectorizer.transform(test_data['lemmatized_text'])
```

[6] Pytho

```python
# Encoding the Labels
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(train_data['label'])
y_test = label_encoder.transform(test_data['label'])
```

[7] Pytho

Model Training and Evaluation

We will train and evaluate three models: Logistic Regression, Random Forest, and Support Vector Machine (SVM), using both feature sets (TF-IDF and Count Vectors) to determine which combination yields the best performance.

```python
# Function to evaluate models
def evaluate_model(model, X_test, y_test, model_name="Model"):
    y_pred = model.predict(X_test)
    print(f"Results for {model_name}:")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Accuracy: {:.2f}\n".format(accuracy_score(y_test, y_pred)))
```

[8] Pytho

```python
# Logistic Regression
```

final_project.ipynb

final_project.ipynb > Final Project > Data Preprocessing > # Tokenization and Lemmatization Function

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· py38_env (Python 3.8.19)

```python
# Function to evaluate models
def evaluate_model(model, X_test, y_test, model_name="Model"):
    y_pred = model.predict(X_test)
    print(f"Results for {model_name}:")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Accuracy: {:.2f}\n".format(accuracy_score(y_test, y_pred)))
```

[8] Python

```python
# Logistic Regression
print("Logistic Regression Performance:")
lr_params = {'C': np.logspace(-3, 3, 10),
             'penalty': ['l1', 'l2'],
             'solver': ['liblinear', 'saga']}

grid_lr_tfidf = GridSearchCV(LogisticRegression(), lr_params, cv=5, scoring='f1_weighted')
grid_lr_tfidf.fit(X_train_tfidf, y_train)
evaluate_model(grid_lr_tfidf.best_estimator_, X_test_tfidf, y_test, "Logistic Regression TF-IDF")

grid_lr_count = GridSearchCV(LogisticRegression(), lr_params, cv=5, scoring='f1_weighted')
grid_lr_count.fit(X_train_count, y_train)
evaluate_model(grid_lr_count.best_estimator_, X_test_count, y_test, "Logistic Regression Count Vectorizer")
```

[9] Python

```
Logistic Regression Performance:
Results for Logistic Regression TF-IDF:
              precision    recall  f1-score   support

           0       0.83      0.93      0.88       789
           1       0.72      0.50      0.59       297

    accuracy                           0.81      1086
   macro avg       0.78      0.71      0.73      1086
weighted avg       0.80      0.81      0.80      1086

Confusion Matrix:
 [[731  58]
 [148 149]]
Accuracy: 0.81

Results for Logistic Regression Count Vectorizer:
              precision    recall  f1-score   support

           0       0.82      0.92      0.87       789
           1       0.69      0.48      0.57       297

    accuracy                           0.80      1086
   macro avg       0.76      0.70      0.72      1086
weighted avg       0.79      0.80      0.79      1086

Confusion Matrix:
 [[724  65]
 [154 143]]
Accuracy: 0.80
```

```python
# Random Forest
print("Random Forest Performance:")
rf_params = {
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [None, 10, 20, 30, 40]
```

final_project.ipynb

final_project.ipynb > Final Project > Data Preprocessing > # Tokenization and Lemmatization Function

Code | Markdown | Run All | Restart | Clear All Outputs | Variables | Outline | ...

py38_env (Python 3.8.19)

```python
# Random Forest
print("Random Forest Performance:")
rf_params = {
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [None, 10, 20, 30, 40]
}

grid_rf_tfidf = GridSearchCV(RandomForestClassifier(), rf_params, cv=5, scoring='f1_weighted')
grid_rf_tfidf.fit(X_train_tfidf, y_train)
evaluate_model(grid_rf_tfidf.best_estimator_, X_test_tfidf, y_test, "Random Forest TF-IDF")

grid_rf_count = GridSearchCV(RandomForestClassifier(), rf_params, cv=5, scoring='f1_weighted')
grid_rf_count.fit(X_train_count, y_train)
evaluate_model(grid_rf_count.best_estimator_, X_test_count, y_test, "Random Forest Count Vectorizer")
```

[10] Python

```
Random Forest Performance:
Results for Random Forest TF-IDF:
              precision    recall  f1-score   support

           0       0.82      0.95      0.88       789
           1       0.78      0.45      0.58       297

    accuracy                           0.82      1086
   macro avg       0.80      0.70      0.73      1086
weighted avg       0.81      0.82      0.80      1086

Confusion Matrix:
 [[752  37]
 [162 135]]
Accuracy: 0.82

Results for Random Forest Count Vectorizer:
              precision    recall  f1-score   support

           0       0.83      0.93      0.87       789
           1       0.71      0.49      0.58       297

    accuracy                           0.81      1086
   macro avg       0.77      0.71      0.73      1086
weighted avg       0.80      0.81      0.80      1086

Confusion Matrix:
 [[730  59]
 [150 147]]
Accuracy: 0.81
```

```python
# Support Vector Machine
print("SVM Performance:")

svm_params = {
    'C': np.logspace(-2, 2, 8),
    'gamma': np.logspace(-2, 2, 8),
    'kernel': ['rbf'],
    'class_weight': ['balanced']
}

grid_svm_tfidf = GridSearchCV(SVC(), svm_params, cv=5, scoring='f1_weighted')
grid_svm_tfidf.fit(X_train_tfidf, y_train)
evaluate_model(grid_svm_tfidf.best_estimator_, X_test_tfidf, y_test, "Optimized SVM TF-IDF")
```

final_project.ipynb

final_project.ipynb > Final Project > Data Preprocessing > # Tokenization and Lemmatization Function

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· py38_env (Python 3.8.19)

```python
# Support Vector Machine
print("SVM Performance:")

svm_params = {
    'C': np.logspace(-2, 2, 8),
    'gamma': np.logspace(-2, 2, 8),
    'kernel': ['rbf'],
    'class_weight': ['balanced']
}

grid_svm_tfidf = GridSearchCV(SVC(), svm_params, cv=5, scoring='f1_weighted')
grid_svm_tfidf.fit(X_train_tfidf, y_train)
evaluate_model(grid_svm_tfidf.best_estimator_, X_test_tfidf, y_test, "Optimized SVM TF-IDF")

grid_svm_count = GridSearchCV(SVC(), svm_params, cv=3, scoring='f1_weighted')
grid_svm_count.fit(X_train_count, y_train)
evaluate_model(grid_svm_count.best_estimator_, X_test_count, y_test, "SVM Count Vectorizer")
```

[11] Python

```
SVM Performance:
Results for Optimized SVM TF-IDF:
              precision    recall  f1-score   support

           0       0.85      0.84      0.85       789
           1       0.59      0.62      0.61       297

    accuracy                           0.78      1086
   macro avg       0.72      0.73      0.73      1086
weighted avg       0.78      0.78      0.78      1086

Confusion Matrix:
 [[662 127]
 [113 184]]
Accuracy: 0.78

Results for SVM Count Vectorizer:
              precision    recall  f1-score   support

           0       0.86      0.84      0.85       789
           1       0.60      0.63      0.61       297

    accuracy                           0.78      1086
   macro avg       0.73      0.73      0.73      1086
weighted avg       0.79      0.78      0.78      1086

Confusion Matrix:
 [[662 127]
 [110 187]]
Accuracy: 0.78
```

```python
# LinearSVC
print("LinearSVC Performance:")
linear_svm_params = {
    'C': np.logspace(-5, 2, 8),
    'loss': ['hinge', 'squared_hinge']
}

grid_linear_svm_tfidf = GridSearchCV(LinearSVC(), linear_svm_params, cv=5, scoring='f1_weighted')
grid_linear_svm_tfidf.fit(X_train_tfidf, y_train)
```

final_project.ipynb

final_project.ipynb > Final Project > Data Preprocessing > # Tokenization and Lemmatization Function

+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ··· py38_env (Python 3.8.19)

```
linear_svm_params = {
    'C': np.logspace(-5, 2, 8),
    'loss': ['hinge', 'squared_hinge']
}

grid_linear_svm_tfidf = GridSearchCV(LinearSVC(), linear_svm_params, cv=5, scoring='f1_weighted')
grid_linear_svm_tfidf.fit(X_train_tfidf, y_train)
evaluate_model(grid_linear_svm_tfidf.best_estimator_, X_test_tfidf, y_test, "LinearSVC TF-IDF")

grid_linear_svm_count = GridSearchCV(LinearSVC(), linear_svm_params, cv=5, scoring='f1_weighted')
grid_linear_svm_count.fit(X_train_count, y_train)
evaluate_model(grid_linear_svm_count.best_estimator_, X_test_count, y_test, "LinearSVC Count Vectorizer")
```

[12] Python

```
LinearSVC Performance:
Results for LinearSVC TF-IDF:
              precision    recall  f1-score   support

           0       0.82      0.95      0.88       789
           1       0.77      0.45      0.57       297

    accuracy                           0.81      1086
   macro avg       0.80      0.70      0.73      1086
weighted avg       0.81      0.81      0.80      1086

Confusion Matrix:
 [[750  39]
 [163 134]]
Accuracy: 0.81

Results for LinearSVC Count Vectorizer:
              precision    recall  f1-score   support

           0       0.83      0.90      0.87       789
           1       0.66      0.53      0.59       297

    accuracy                           0.80      1086
   macro avg       0.75      0.71      0.73      1086
weighted avg       0.79      0.80      0.79      1086

Confusion Matrix:
 [[710  79]
 [141 156]]
Accuracy: 0.80
```

Experimental results

Please organize your results similar as the following table (you can choose other ways to display your table)

	Sexist			Not Sexist			weighted average		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
TF-IDF + Logistic Regression	0.72	0.5	0.59	0.83	0.93	0.88	0.8	0.81	0.8
Count Vec + Logistic Regression	0.69	0.48	0.57	0.82	0.92	0.87	0.79	0.8	0.79
TF-IDF + Random Forest	0.78	0.45	0.58	0.82	0.95	0.88	0.81	0.82	0.8
Count Vec + Random Forest	0.71	0.49	0.58	0.83	0.93	0.87	0.8	0.81	0.8
TF-IDF + SVM	0.59	0.62	0.61	0.85	0.84	0.85	0.78	0.78	0.78
Count Vec + SVM	0.6	0.63	0.61	0.86	0.84	0.85	0.79	0.78	0.78
TF-IDF + LinearSVC	0.77	0.45	0.57	0.82	0.95	0.88	0.81	0.81	0.8
Count Vec + LinearSVC	0.66	0.53	0.59	0.83	0.9	0.87	0.79	0.8	0.79

Summary

Summary

1. What preprocessing steps do you follow?

Answer: Preprocessing involved several steps to clean and standardize the input text data:

- **URLs and User Tags Removal**: Stripping out hyperlinks and user mentions, which are irrelevant to text classification.
- **Punctuation and Numbers Removal**: Deleting punctuation and numerical figures to focus on textual content.
- **Lowercasing All Text**: Standardizing all text to lower case to avoid duplication of tokens based on case differences.
- **Tokenization**: Breaking down text into individual words or tokens.
- **Lemmatization with POS Tagging**: Converting words to their base form using part-of-speech tagging to accurately derive the lemma.
- **Stop Words Removal**: Eliminating commonly used words (such as "the", "a", "in") that do not contribute significantly to the meaning of the text.

2. How do you select the features from the inputs?

Answer: Features were selected using two primary text representation techniques:

- **TF-IDF Vectorization**: This method weighs words based on their frequency and relevance across documents, helping to elevate words that are unique to a document's context.
- **Count Vectorization**: This approach counts the occurrences of each word in the text to form feature vectors.

3. Which model you use and what is the structure of your model?

Answer: The models tested included:

- **Random Forest**: An ensemble model using multiple decision trees to improve classification accuracy and control over-fitting.
- **Support Vector Machine (SVM)**: Employs kernels to handle nonlinear data separation effectively.
- **LinearSVC**: A linear type of SVM that is faster on large datasets and effective with high dimensional spaces. Each model was configured with hyperparameters optimized through cross-validation using `GridSearchCV`.

4. How do you train your model?

Answer: Training involved the following steps:

- **Splitting Data**: Dividing data into training and testing sets to ensure the model is validated on unseen data.
- **Applying Vectorizers**: Transforming text data into numerical formats using TF-IDF and Count Vectorizers.
- **Grid Search CV**: Utilizing GridSearchCV to search through predefined hyperparameter spaces to find the optimal settings for each model.
- **Model Fitting**: Training each model on the vectorized training data and tuning them based on the validation results from the cross-validation process.

5. What is the performance of your best model?

Answer: The **Random Forest with TF-IDF Vectorizer** has shown to be the best performing model according to the metrics evaluated on the test set:

- **Accuracy**: 82%
- **Precision**: 81% (weighted average)
- **Recall**: 82% (weighted average)
- **F1-Score**: 80% (weighted average)
- **Confusion Matrix**: True Negative = 752, False Positive = 37, False Negative = 162, True Positive = 135

6. What other models or feature engineering methods would you like to implement in the future?

Answer: Future directions could include:

- **Enhanced Feature Engineering**: Incorporating Word2Vec or GloVe to potentially improve model performance.
- **Hyperparameter Optimization**: Further refining models through more sophisticated hyperparameter optimization techniques.