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 (https://codalab.lisn.upsaclay.fr/competitions/7124#learn_the_details-overview) you can find a detailed introduction to this task.

In [2]:
```
# Imports
import warnings
# Suppress warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_ma

import re
import nltk
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import string

from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize

from sklearn.svm import SVC
from sklearn.linear_model import RidgeClassifier

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import precision_score, accuracy_score, f1_score, precisi
```

LOADING IN THE DATA

In [ ]:
```
# Text Loading

# read in the data
message_df = pd.read_csv('edos_labelled_data.csv')

# split the data into train and test
train_data = message_df[message_df['split'] == 'train']
test_data = message_df[message_df['split'] == 'test']

# store train x and y data
X_train = train_data['text']
y_train = train_data['label']

# store test x and y data
X_test = test_data['text']
y_test = test_data['label']
```

TEXT PREPROCESSING STEPS

In [287]:
```
# Text Preprocessing

# initalize stemmer and lemmatizer
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# initialize stopwords for the english set
stop_words = set(stopwords.words('english'))

# function to preprocess the text data
def preprocess_text(text):
    # set to lowercase
    text = text.lower()

    # substitutes all non-letters and whitespaces with an empty character
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # remove punctuation
    text = ''.join([word for word in text if word not in string.punctuation])

    # tokenize the text
    tokens = word_tokenize(text)

    # utilize stop words
    stop_removed_words = [word for word in tokens if word not in stop_words]

    # lemmatize and stem
    lemmatized_words = [lemmatizer.lemmatize(word) for word in stop_removed_wor
    stemmed_words = [stemmer.stem(word) for word in lemmatized_words]

    # return the processed text
    preprocessed_text = ' '.join(stemmed_words)
    return preprocessed_text

# apply pre-processing to both the train set and the test set
X_train = X_train.apply(preprocess_text)
X_test = X_test.apply(preprocess_text)
```

ENCODING METHODS

In [288]:
```
# TF-IDF Encoding Method

# initialize the tfidf vectorizer
tfidf_vectorizer = TfidfVectorizer()

# vectorize both the train and test set
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

In [289]:
```
# Word2Vec Encoding Method

# tokenize the text
split_text = [sentence.split() for sentence in X_train]

# initialize a w2v model
w2v_model = Word2Vec(sentences=split_text, vector_size=100, window=5, min_count

# method to vectorize each sentence
def vectorize(sentence):
    words = sentence.split()
    vecs = [w2v_model.wv[word] for word in words if word in w2v_model.wv]
    if vecs:
        return np.mean(vecs, axis=0)
    return np.zeros(100)

# apply the w2v encoding on both the train and test set
X_train_w2v = np.array([vectorize(sentence) for sentence in X_train])
X_test_w2v = np.array([vectorize(sentence) for sentence in X_test])
```

LOGISTIC REGRESSION MODELS

In [290]:
```
# Logistic Regression Model with TFIDF

# initialize a logistic regressoin model
logreg_model = LogisticRegression(max_iter=1000)

# initialize hyperparameters
param_grid = {
    'C': [0.1, 1, 10, 100],
    'solver': ['liblinear', 'lbfgs']
}

# find the most optimal hyperparameters using grid search
grid_search = GridSearchCV(logreg_model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_tfidf, y_train)

# prin the best parameters
best_params = grid_search.best_params_
print("Best Parameters: ")
print(best_params)

# set the best estimator
best_log_reg = grid_search.best_estimator_

# logreg_model.fit(X_train_tfidf, y_train)

# predict the data using the most optimal hyperparameters
y_pred = best_log_reg.predict(X_test_tfidf)

# print a classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# store aspects of the classification report for usage in the evaluation step
tfidf_log_precision, tfidf_log_recall, tfidf_log_f1_score, tfidf_log_support =
tfidf_log_w_precision, tfidf_log_w_recall, tfidf_log_w_f1_score, _ = precision_
```

```
Best Parameters:
{'C': 10, 'solver': 'liblinear'}

Classification Report:
              precision    recall  f1-score   support

  not sexist       0.81      0.91      0.86       789
      sexist       0.65      0.45      0.53       297

    accuracy                           0.78      1086
   macro avg       0.73      0.68      0.69      1086
weighted avg       0.77      0.78      0.77      1086
```

In [291]:
```
# Logistic Regression Model with Word2Vec

# initialize a logistic regressoin model
logreg_model = LogisticRegression(max_iter=1000)

# initialize hyperparameters
param_grid = {
    'C': [0.1, 1, 10, 100],
    'solver': ['liblinear', 'lbfgs']
}

# find the most optimal hyperparameters using grid search
grid_search = GridSearchCV(logreg_model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_w2v, y_train)

# prin the best parameters
best_params = grid_search.best_params_
print("Best Parameters: ")
print(best_params)

# set the best estimator
best_log_reg = grid_search.best_estimator_

# logreg_model.fit(X_train_tfidf, y_train)

# predict the data using the most optimal hyperparameters
y_pred = best_log_reg.predict(X_test_w2v)

# print a classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# store aspects of the classification report for usage in the evaluation step
w2v_log_precision, w2v_log_recall, w2v_log_f1_score, w2v_log_support = precisio
w2v_log_w_precision, w2v_log_w_recall, w2v_log_w_f1_score, _ = precision_recall
```

```
Best Parameters:
{'C': 1, 'solver': 'liblinear'}

Classification Report:
              precision    recall  f1-score   support

  not sexist       0.77      0.95      0.85       789
      sexist       0.63      0.23      0.33       297

    accuracy                           0.75      1086
   macro avg       0.70      0.59      0.59      1086
weighted avg       0.73      0.75      0.71      1086
```

SVM MODELS

In [292]:
```
# SVM Model with TFIDF

# initalize the SVM Model
best_svm = SVC(kernel='linear', C=1)

# initialize hyperparameters
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly']
}

# find most optimal hyperparameters using grid search
# grid_search = GridSearchCV(svm_model, param_grid, cv=5, scoring='accuracy')
# grid_search.fit(X_train_tfidf, y_train)

# set the best estimator
# best_svm = grid_search.best_estimator_

# print the best parameters
print("Best Parameters from previous runs are: Kernal = linear, C = 1. It takes

# fit the SVM model
best_svm.fit(X_train_tfidf, y_train)

# use the SVM model to predict on the test set
y_pred = best_svm.predict(X_test_tfidf)

# print a classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# store aspects of the classification report for usage in the evaluation step
tfidf_svm_precision, tfidf_svm_recall, tfidf_svm_f1_score, tfidf_svm_support =
tfidf_svm_w_precision, tfidf_svm_w_recall, tfidf_svm_w_f1_score, _ = precision_
```

```
Best Parameters from previous runs are: Kernal = linear, C = 1. It takes too
long to run if I do it every time.

Classification Report:
              precision    recall  f1-score   support

  not sexist       0.82      0.95      0.88       789
      sexist       0.75      0.43      0.55       297

    accuracy                           0.81      1086
   macro avg       0.78      0.69      0.71      1086
weighted avg       0.80      0.81      0.79      1086
```

In [293]:
```
# SVM Model with Word2Vec

# initalize the SVM Model
svm_model = SVC(kernel='rbf')

# initialize hyperparameters
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly']
}

# find most optimal hyperparameters using grid search
# grid_search = GridSearchCV(svm_model, param_grid, cv=5, scoring='accuracy')
# grid_search.fit(X_train_tfidf, y_train)

# set the best estimator
# best_svm = grid_search.best_estimator_

# print the best parameters
print("Best Parameters from previous runs are: Kernal = rbf. It takes too long

# fit the SVM model
svm_model.fit(X_train_w2v, y_train)

# use the SVM model to predict on the test set
y_pred = svm_model.predict(X_test_w2v)

# print a classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# store aspects of the classification report for usage in the evaluation step
w2v_svm_precision, w2v_svm_recall, w2v_svm_f1_score, w2v_svm_support = precisio
w2v_svm_w_precision, w2v_svm_w_recall, w2v_svm_w_f1_score, _ = precision_recall
```

```
Best Parameters from previous runs are: Kernal = rbf. It takes too long to ru
n if I do it every time.

Classification Report:
              precision    recall  f1-score   support

  not sexist       0.78      0.96      0.86       789
      sexist       0.72      0.26      0.39       297

    accuracy                           0.77      1086
   macro avg       0.75      0.61      0.62      1086
weighted avg       0.76      0.77      0.73      1086
```

RANDOM FOREST MODELS

In [294]:
```
# Random Forest with TFIDF

# I attempted to optimize hyperparameters, however it was taking an extremely l

# initialize random forest
random_forest = RandomForestClassifier()

# fit the random forest
random_forest.fit(X_train_tfidf, y_train)

# predict using the random forest
y_pred_rf = random_forest.predict(X_test_tfidf)

# print a classification report
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))

# store aspects of the classification report for usage in the evaluation step
tfidf_rf_precision, tfidf_rf_recall, tfidf_rf_f1_score, tfidf_rf_support = pre
tfidf_rf_w_precision, tfidf_rf_w_recall, tfidf_rf_w_f1_score, _ = precision_re
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

  not sexist       0.82      0.98      0.89       789
      sexist       0.91      0.42      0.57       297

    accuracy                           0.83      1086
   macro avg       0.86      0.70      0.73      1086
weighted avg       0.84      0.83      0.81      1086
```

In [295]:
```
# Random Forest with W2V

# initialize random forest
random_forest = RandomForestClassifier()

# fit the random forest
random_forest.fit(X_train_w2v, y_train)

# predict using the random forest
y_pred_rf = random_forest.predict(X_test_w2v)

# print a classification report
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))

# store aspects of the classification report for usage in the evaluation step
w2v_rf_precision, w2v_rf_recall, w2v_rf_f1_score, w2v_rf_support = precision_re
w2v_rf_w_precision, w2v_rf_w_recall, w2v_rf_w_f1_score, _ = precision_recall_fs
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

  not sexist       0.76      0.96      0.85       789
      sexist       0.65      0.22      0.32       297

    accuracy                           0.75      1086
   macro avg       0.71      0.59      0.59      1086
weighted avg       0.73      0.75      0.71      1086
```

RIDGE REGRESSION MODELS

In [296]:
```
# Ridge Regression Model with TFIDF
# not included in evaluation

# initialize a ridge regression model and fit it
ridge_model = RidgeClassifier()
ridge_model.fit(X_train_tfidf, y_train)

# use the ridge regression to predict
y_pred_ridge = ridge_model.predict(X_test_tfidf)

# store aspects of the classification report for usage in the evaluation step
print("\nRidge Regression Classification Report:")
print(classification_report(y_test, y_pred_ridge))
```

```
Ridge Regression Classification Report:
              precision    recall  f1-score   support

  not sexist       0.81      0.93      0.87       789
      sexist       0.71      0.43      0.54       297

    accuracy                           0.80      1086
   macro avg       0.76      0.68      0.70      1086
weighted avg       0.78      0.80      0.78      1086
```

In [297]:
```
# Ridge Regression Model with Word2Vec
# not included in evaluation

# initialize a ridge regression model and fit it
ridge_model = RidgeClassifier()
ridge_model.fit(X_train_w2v, y_train)

# use the ridge regression to predict
y_pred_ridge = ridge_model.predict(X_test_w2v)

# store aspects of the classification report for usage in the evaluation step
print("\nRidge Regression Classification Report:")
print(classification_report(y_test, y_pred_ridge))
```

```
Ridge Regression Classification Report:
              precision    recall  f1-score   support

  not sexist       0.76      0.95      0.85       789
      sexist       0.63      0.22      0.33       297

    accuracy                           0.75      1086
   macro avg       0.70      0.59      0.59      1086
weighted avg       0.73      0.75      0.71      1086
```

EVALUATION

In [298]:
```
# Evaluation

print("Appreviations: S = sexist, NS = non-sexist, WA = weighted average\n")

# set up the data in the table with proper labels
data = {
    'Feature + Model': ['TF-IDF + Logistic Regression', 'W2V + Logistic Regress
                        'TF-IDF + SVM', 'W2V + SVM',
                        'TF-IDF + Random Forest', 'W2V + Random Forest'],
    'S Precision':     [tfidf_log_precision[1], w2v_log_precision[1], tfidf_svr
                            w2v_svm_precision[1], tfidf_rf_precision[1], w2v_r
    'S Recall':      [tfidf_log_recall[1], w2v_log_recall[1], tfidf_svm_recall
                            w2v_svm_recall[1], tfidf_rf_recall[1], w2v_rf_recal
    'S F1-score':      [tfidf_log_f1_score[1], w2v_log_f1_score[1], tfidf_svm_
                            w2v_svm_f1_score[1], tfidf_rf_f1_score[1], w2v_rf_

    'NS Precision': [tfidf_log_precision[0], w2v_log_precision[0], tfidf_svm_pr
                            w2v_svm_precision[0], tfidf_rf_precision[0], w2v_r
    'NS Recall':  [tfidf_log_recall[1], w2v_log_recall[0], tfidf_svm_recall[0],
                            w2v_svm_recall[0], tfidf_rf_recall[0], w2v_rf_recal
    'NS F1-score':  [tfidf_log_f1_score[0], w2v_log_f1_score[0], tfidf_svm_f1_s
                            w2v_svm_f1_score[0], tfidf_rf_f1_score[0], w2v_rf_

    'WA Precision': [tfidf_log_w_precision, w2v_log_w_precision, tfidf_svm_w_pr
                            w2v_svm_w_precision, tfidf_rf_w_precision, w2v_rf_w
    'WA Recall':  [tfidf_log_w_recall, w2v_log_w_recall, tfidf_svm_w_recall,
                            w2v_svm_w_recall, tfidf_rf_w_recall, w2v_rf_w_recal
    'WA F1-score':  [tfidf_log_w_f1_score, w2v_log_w_f1_score, tfidf_svm_w_f1_s
                            w2v_svm_w_f1_score, tfidf_rf_w_f1_score, w2v_rf_w_
}

# make the data into dataframe
df = pd.DataFrame(data)

# print the dataframe
print(df)

# determine best performing model based on weighted F1 Score
best_performance = df[df['WA F1-score'] == df['WA F1-score'].max()]

# print the best performing model
print("\nBest Performing Model:")
print(best_performance)
```

```
Appreviations: S = sexist, NS = non-sexist, WA = weighted average

                Feature + Model  S Precision  S Recall  S F1-score  \
0  TF-IDF + Logistic Regression     0.648780  0.447811    0.529880
1     W2V + Logistic Regression     0.632075  0.225589    0.332506
2                 TF-IDF + SVM     0.752941  0.430976    0.548180
3                    W2V + SVM     0.722222  0.262626    0.385185
4        TF-IDF + Random Forest     0.905109  0.417508    0.571429
5           W2V + Random Forest     0.653061  0.215488    0.324051

   NS Precision  NS Recall  NS F1-score  WA Precision  WA Recall  WA F1-score
0      0.813848   0.447811     0.858683      0.768705   0.782689     0.768762
1      0.765306   0.950570     0.847937      0.728870   0.752302     0.706976
2      0.815502   0.946768     0.876246      0.798393   0.805709     0.786527
3      0.776074   0.961977     0.859083      0.761346   0.770718     0.729481
4      0.817703   0.983523     0.892980      0.841607   0.828729     0.805042
5      0.764170   0.956907     0.849747      0.733784   0.754144     0.705979

Best Performing Model:
          Feature + Model  S Precision  S Recall  S F1-score  NS Precision  \
4  TF-IDF + Random Forest     0.905109  0.417508    0.571429      0.817703

   NS Recall  NS F1-score  WA Precision  WA Recall  WA F1-score
4   0.983523      0.89298      0.841607   0.828729     0.805042
```

Summary

1. What preprocessing steps do you follow?

   Your answer: For preprocessing steps, I first convert the text to lowercase in order to make sure there is uniformity in the data. Then, I remove non-alphabetic characters as I found that my models performed better without them included within the data, as well as removing punctuation. I then break the text into tokens and remove common stop words using the NLTK stopwords list. Finally, I lemmatize the words, reducing them to their dictionary form, and then stem them.
2. How do you select the features from the inputs?

   Your answer: The two methods I used to select features were TF-IDF and Word2Vec. For TF-IDF, I run the vectorizer to convert the raw text data into numerical features. For Word2Vec, I tokenize the text, splitting it into a list of words. Then I embed the words with Word2Vec using those tokenized sentences. I specified the parameters to those which I found gave the most optimal results. I then vectorize the sentences, and then perform this vectorization to fit the train data and transform the test data to ensure that there is no test data leaked into the training process. Overall, I noticed Word2Vec was giving me consistently worse results on average then TF_IDF, so all of my best models are with TF-IDF.
3. Which model you use and what is the structure of your model?

   Your answer: The three models I selected were Logisitic Regression, Support Vector Machine, and Random Forest. I also trained a Ridge Regression model, but the performance was not much better than the random forest and logistic regression models. I

decided not to include the results in the evaluation so I could keep the table more concise, however the classfication reports and code are still there.

For the logistic regression model, the structure is optimized using hyperparameters, with the 'C' parameter being particularly important for achieving the highest possible accuracy. I also optimized the 'solver' parameter.

For the support vector machine model, the structure uses a linear kernal due to giving the best performance amongst other kernals.

For the random forest model, the structure uses the default for random forest, since I found this not only gave the best performance, but attempting optimize hyperparameters was struggling to run on my computer.

Overall, random forest ran the best with TF-IDF when compared with the other models and feature selection methods.

4. How do you train your model?

   Your answer: All three models are trained using the training set with each of the two feature selection methods (TF-IDF and Word2Vec):

   For the logistic regression model, it is trained by fitting the data with different combinations of hyperparameters to find the best model to be selected. The best estimator is then chosen based on the scoring method that I specified. It then makes its prediction on the test set.

   For the support vector machine model, it trained by fitting the data in a linear kernal and then predicting on the test set.

   For the random forest model, it is trained similarly to the other two models.

5. What is the performance of your best model?

   Your answer: The best performing model was TF-IDF + Random Forest, which gave me the highest Weighted Average F1-Score as shown above in the evaluation section. It shows higher precision for the 'not sexist' class than the 'sexist' class, which was typical between all of my models. Since 'not sexist' has a high recall with this model, it indicates that it correctly identifies a majority of those cases.

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

   Your answer: I would have liked to try using a pre-trained BERT-based model, specifically DeBERTaV3, however I was not able to figure out how to get it reliably working and I found it to be potentially too advanced for my current understanding of these topics. From my research into optimal models and methods to use, I found that it was a popular choice and likely could have given better results than the ones that I found using my models. I was really curious to see how something more advanced would perform on this data. It would also be interesting how other models would have done with the data, such as a neural network or more complex options which may be able to capture necessary information