Retrieval and Preprocessing

The imported libraries include pandas, numpy, sklearn, nltk, string, contractions, and Levenshtein. The datasets were converted from the text files into pandas DataFrames for easier data preprocessing and manipulation.

In preprocessing, the special apostrophe character (U+2019) was replaced with the more general apostrophe character (U+0027). The contractions.fix function was used to expand any contractions (e.g. "they're" becomes "they are"). Afterwards, any punctuation was removed by utilizing string.punctuation from the string library, and sentences were converted to lowercase.

Feature Definition

For features, the following was defined:

- **Euclidean distance (edist)** - A TfidfVectorizer from sklearn was used to convert the sentences into vectors. The euclidean_distances function from sklearn was used to get the Euclidean distance.
- **Cosine similarity (csim)** - Similarly, a TfidfVectorizer and the cosine_similarity function from sklearn was used to get the cosine similarity.
- **Number of words in the sentences (length1, length2)** - The number of words was found by getting the length of the array returned by .split() method applied to each sentence.
- **Difference in the number of words (lengthdiff)** - The absolute value of (length1 – length2).
- **Number of overlapping words (slices)** - A for loop was used to iterate through each sentences and count the number of matching words.
- **Meteor score (mscores)** – The meteor_score function from nltk was used to get the meteor score.
- **Levenshtein distance (ldist)** - The distance function from Levenshtein was used to get the Levenshtein distance.
- **Levenshtein ratio (lratio)** - The ratio function from Levenshtein was used to get the Levenshtein ratio.
- **BLEU-1, BLEU-2, BLEU-3 (bleu1, bleu2, bleu3)** - The sentence_bleu function from nltk were used to get the BLEU score. Weights were defined to specify up to how many n-grams were to be used for each BLEU score (e.g. two weights were defined to get BLEU-2).

Model

The model used was a Support Vector Classifier using a 3rd-degree polynomial kernel function with a kernel coefficient (gamma) of 0.1, a regularization parameter (C) of 10, and an independent term of 10 (coef0). The class_weight parameter was set to "balanced" to account for the class imbalance between positives and negatives in the training data.

The model was put through a pipeline that used a StandardScaler with its with_mean parameter set to "False", meaning the data was not centered before scaling.

The final accuracy achieved on the dev set with this model is 0.755524861878453.

Additional Comments

A CountVectorizer was initially used to vectorize the sentences to calculate Euclidean and cosine similarity, but TdidfVectorizer achieved a better accuracy.

BLEU-4 was also calculated and included in the features, but it worsened accuracy, so it was omitted.

Other kernel functions such as linear and rbf were used as well as a classifier using logistic regression, but they did not perform as well as the polynomial SVC.

Other scalers were used such as MinMaxScaler and MaxAbsScaler, but they did not improve accuracy.

Throughout the process, I learned that bagging was not useful for the SVC with a polynomial kernel, but it would sometimes improve accuracy for the SVC with a rbf kernel.

Similarly, PCA almost never improved accuracy except for one time with the SVC with a rbf kernel.

Also, there were no cases were boosting improved accuracy (GradBoost and AdaBoost were attempted).

Interestingly, adding stopwords worsened accuracy.

Another specific phenomenon I noticed was that SVCs with different kernel functions and different parameters can yield the same exact accuracy.