**Name:** Nooreldean Koteb

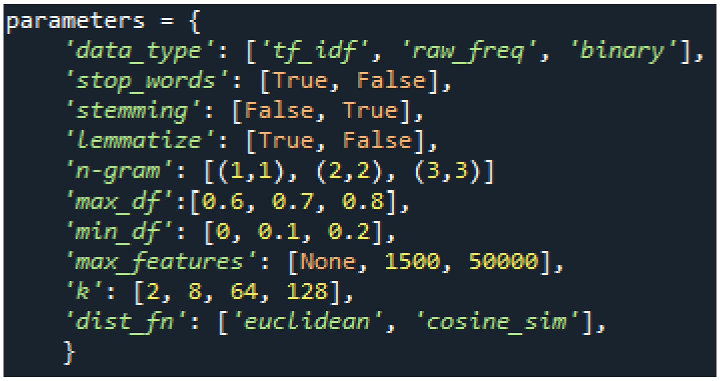
**Miner Username:** IcyEagle

**Rank:** 116 **Accuracy:** 50%

**Instructions:**

My code is compartmentalized into multiple functions and helper functions. The main 2 functions that are used to run my program are cross_val() and final_predict(). Both functions call the preprocess function that automatically preprocesses the code and delivers it to be tested.

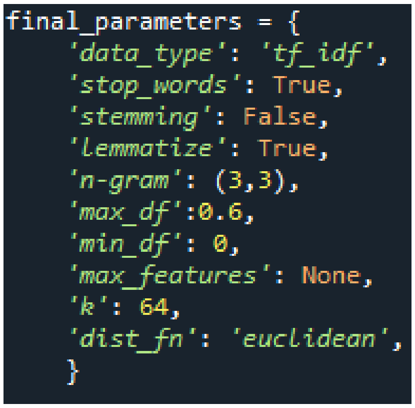
The cross-validation function cross_val(train_dat, parameters, reduce=None) takes in 3 parameters: a string with the location of the training data file, a dictionary of parameters to test and optionally a reduce int to reduce the amount of data used. The parameters dictionary should be formatted like the image to the right. The cross_val function will then make combinations of these parameters and automatically throw out cases where stemming and lemmatization are both set to True.



```
parameters = {
    'data_type': ['tf_idf', 'raw_freq', 'binary'],
    'stop_words': [True, False],
    'stemming': [False, True],
    'lemmatize': [True, False],
    'n-gram': [(1,1), (2,2), (3,3)]
    'max_df':[0.6, 0.7, 0.8],
    'min_df': [0, 0.1, 0.2],
    'max_features': [None, 1500, 50000],
    'k': [2, 8, 64, 128],
    'dist_fn': ['euclidean', 'cosine_sim'],
    }
```

The cross_val() function will return a dictionary with all the results. The results are then sorted, and 2 files are written “results.txt” with all results pre-sort, and “sorted_results.txt” with the sorted results by accuracy.

The final_predict(train_dat, test_dat, param, reduce=None) function is used to predict the final results using the “test.dat” file. This function is very similar to cross_val, except it takes in an additional parameter test_dat which is a string with the location of the test.dat file. The parameters for final_predict are also slightly different taking only 1 value per category. The final_predict () function writes to a new file “final_results.txt” with its predictions as +1 or -1, and the function returns a list of all predictions.



```
final_parameters = {
    'data_type': 'tf_idf',
    'stop_words': True,
    'stemming': False,
    'lemmatize': True,
    'n-gram': (3,3),
    'max_df':0.6,
    'min_df': 0,
    'max_features': None,
    'k': 64,
    'dist_fn': 'euclidean',
    }
```

**Approach:**

I began cross validation by testing all possible combinations for all categories of the listed parameters in the pictures above. I then realized that cosine similarity consistently gave me a lower score, and that lemmatization and removing stop words consistently did better than when they were not True. These original tests were done on a reduced data set of 12,000 reviews from the training data.

I then started testing various max_df, min_df, max_features, and k values. I realized that the best results came with lower min_df and a max_df around 0.6 and 0.8. max_features did not seem to make an impact, and k values seemed best at higher values between 128 and 512. TFIDF also did the best of the 3 data representations. The best results I got from this were high 60s to low 70s with the highest being 72% on a full training set with a 70-30 split.

72% Accuracy: {'data_type': 'tf_idf', 'stop_words': True, 'stemming': False, 'lemmatize': True, 'max_df': 0.8, 'min_df': 0, 'max_features': None, 'k': 512, 'dist_fn': 'euclidean'}

I then decided to test n-gram values to try and improve my accuracy. I found that n-gram range (3,3) did the best. I then started multiple cross validation tests to find the best parameters around this new n-gram range and found that a min_df of 0, max_df of 0.6, max_features None, and a much lower k value of 64 was the best performing giving me an accuracy of 78.5% on a full training data set with a 70-30 split.

78.5% Accuracy after n-gram change to 3: {'data_type': 'tf_idf', 'stop_words': True, 'stemming': False, 'lemmatize': True, 'max_df': 0.6, 'min_df': 0, 'max_features': None, 'k': 64, 'dist_fn': 'euclidean'}

I ran cross validation tests with different data representations in 3 command prompts to get faster results. Due to the number of tests that I've run and the various parameters I've tried it would be difficult to create a table that would fit in this report. I have however included all my cross-validation logs in a folder named cross validation results next to this report. Each test prints the parameters, the confusion matrix, f1-score, and accuracy. These are html files that can be viewed in any browser.

**Efficiency:**

I tried to make my code as efficient as possible, it takes 45 seconds to 2 minutes to read and preprocess data, and 3-1hr minutes to run a test depending on its parameters. One of the first things I did to cut down on preprocessing time was combine stop word removal with stemming and lemmatization. I also used the re library to remove numbers and punctuation quickly from reviews.

I had implemented my own euclidean distance and cosine similarity functions, but they were taking too long so I substituted them for the sklearn pairwise version. I also used the sklearn tfidf vectorizer instead the one I implemented because it was faster with more options to tune.

Some of the things I tuned were max_df, min_df, and max_features. Max_features did not seem to have much of an impact overall however it did slightly lower results and make tests faster. max_df was the most effective in increasing speed and giving better results.

**Libraries:**

I used nltk for text processing I imported porter stemmer , WordNetLemmatizer and stopwords to stem, lemmatize and remove stopwords I also imported word_tokenize to help in text preprocessing. I also used re for more feature reduction to remove numbers and punctuation. I imported accuracy_score, f1_Score, and confusion_matrix for cross validation evaluation. I also imported train_test_Split to split the training data for cross validation randomly. I imported product from itertools to create a combination of all possible parameters. I also imported Tfidfvectorizer and countvectorizer for data representation. I did implement a tfidf function in my code however the sklearn one was much faster with more options that is why I decided to use that instead. I also imported standardScaler from sklearn to scale tfidf matrices.