Text Classification

STEP - 1: Extraction of Data

```
import numpy as np
import pandas as pd
from sklearn import feature_extraction, linear_model, model_selection, preprocessing
import pandas as pd

df1 = pd.read_csv('/kaggle/input/wn23-data/WN23_PA_training_tweets.txt', delimiter=',',
encoding='iso-8859-1')
df2 = pd.read_csv('/kaggle/input/wn23-data/WN23_PA_training_labels.txt', delimiter=',',
encoding='iso-8859-1')

train_df = pd.merge(df1, df2, on='TweetID')
test_df = pd.read_csv('/kaggle/input/wn23-data/WN23_PA_test_tweets.txt', delimiter=',',
encoding='iso-8859-1')
```

The code snippet above imports necessary packages, reads in training and testing data from files, and merges the training data with its corresponding labels.

The code begins by importing numpy, pandas, and several modules from scikit-learn. Numpy is a package for scientific computing with Python, and pandas is a library used for data manipulation and analysis. Scikit-learn is a machine learning library for Python that provides a range of tools for building predictive models.

Next, the training and testing data are read in from files using pandas' read_csv function. The delimiter parameter specifies that the fields in the file are separated by commas, and the encoding parameter specifies the character encoding used in the file.

The training data is split across two files: WN23_PA_training_tweets.txt, which contains the tweets, and WN23_PA_training_labels.txt, which contains the labels for the tweets. The pd.merge function is used to merge the two dataframes based on a common column, TweetID, so that each tweet is matched with its corresponding label.

Finally, the testing data is read in from WN23_PA_test_tweets.txt, which contains only the tweets themselves. This data will be used to test the performance of the trained model. Overall, this code snippet provides a good starting point for analyzing and modeling the data provided in these files, using Python and scikit-learn.

STEP - 2: Preprocessing Data

Remove unnecessary symbols, stop words, Tokenize, Lemmatize, Stemming every word

```
def process_tweet(tweet):
    stemmer = PorterStemmer()
    stopwords_english = stopwords.words('english')
    tweet = re.sub(r'\$\w*', '', tweet)
    tweet = re.sub(r'^RT[\s]+', '', tweet)
    tweet = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet)
    tweet = re.sub(r'#', '', tweet)
    tweet = re.sub(r'[^\x00-\x7F]+', ' ', tweet)
    tweet = re.sub(r'\W+', ' ', tweet)
    tweet = re.sub(r'\d+', '', tweet)
    tweet = tweet.replace("_", " ")
    # tokenize tweets
    tokenizer = TweetTokenizer(preserve_case=True, strip_handles=True,
                               reduce_len=True)
    tweet_tokens = tokenizer.tokenize(tweet)
    tweets_clean = []
    for word in tweet_tokens:
        if (word not in stopwords_english and  # remove stopwords
                word not in string.punctuation):  # remove punctuation
            # tweets_clean.append(word)
            stem_word = stemmer.stem(word)  # stemming word
            tweets_clean.append(stem_word)
    return tweets_clean
```

This code defines a function process_tweet that takes a tweet as input and performs several text preprocessing steps using various NLP tools from the NLTK libraries.

First, the code imports several necessary modules, including NumPy, regular expressions (re), string, the WordNetLemmatizer and PorterStemmer classes from NLTK, the stopwords corpus from NLTK, the TweetTokenizer from NLTK.

The function takes a tweet as input, and then performs the following text preprocessing steps:

- Remove stock market tickers (e.g. $AAPL)
- Remove retweets (e.g. "RT")
- Remove hyperlinks
- Remove hashtags (#)
- Replace non-ASCII characters with spaces
- Remove all non-alphanumeric characters except underscores (_)
- Remove all digits

- Replace underscores with spaces
- Tokenize the tweet using the TweetTokenizer from NLTK
- Remove stopwords and punctuation marks
- Stem each word using the PorterStemmer from NLTK

The resulting list of cleaned and stemmed words is returned as output.

This function can be used to preprocess tweets before further analysis, such as sentiment analysis or topic modeling. It demonstrates how a combination of different NLP techniques can be used to effectively clean and normalize social media text data.

Adding text sentiment column using TextBlob to every tweet based on hashtags in the tweet

```
import pandas as pd
import re
from textblob import TextBlob

data_df = train_df
data_df['hashtags'] = data_df['Tweet'].apply(lambda x: re.findall(r'\#\w+', x))
data_df['hashtags'] = data_df['hashtags'].apply(lambda x: [re.sub(r'[^\w\s]','',h[1:]) for h in x])
data_df['hashtags_sentiment'] = data_df['hashtags'].apply(lambda x: [TextBlob(h).sentiment.polarity for h in x])
data_df['text_sentiment'] = data_df['Tweet'].apply(lambda x: TextBlob(x).sentiment.polarity)
```

Now we perform text preprocessing and sentiment analysis on a dataset of tweets, using Python and the TextBlob library.
First, the training data (train_df) is loaded into a pandas DataFrame called data_df. Next, the code extracts hashtags from the 'Tweet' column and adds them to a new 'hashtags' column. This is done using a regular expression that matches any word preceded by a pound sign (#). The re.findall function is used to find all matches in each tweet.
After extracting the hashtags, the code removes the pound sign and any punctuation marks from them. This is done using a lambda function that applies the re.sub function to each hashtag in the 'hashtags' column.
The sentiment of each hashtag is then analyzed using TextBlob's sentiment.polarity function, which returns a score between -1 and 1 indicating the sentiment of the text (negative to positive). The scores for each hashtag are added to a new 'hashtags_sentiment' column using another lambda function.
Finally, the sentiment of the entire tweet is analyzed using TextBlob's sentiment.polarity function, and the score is added to a new 'text_sentiment' column using another lambda function..
Overall, this code demonstrates some basic text preprocessing and sentiment analysis techniques that can be applied to social media data. It can be used as a starting point for more advanced natural language processing tasks, such as sentiment analysis on longer text documents.

cls.Tweet = train_df.Tweet.apply(lambda x: process_tweet(x))

This code applies the process_tweet function mentioned above to each tweet in the train_df DataFrame and replaces the original tweet text with the cleaned and stemmed version of the tweet.
The resulting cleaned and stemmed tweet is returned for each element, and the apply method replaces the original tweet text with the cleaned and stemmed version.

```
from tqdm import tqdm

tokens_list = []
for i in tqdm(range(len(cls))):
    tokens = process_tweet(cls['Tweet'].iloc[i])
    tokens_list.extend(tokens)
word_dictionary = set(tokens_list)
print(len(word_dictionary))
```

```
100%|██████████| 4000/4000 [00:01<00:00, 2064.12it/s]
7953
```

The tqdm module is used to display a progress bar for the loop over all the tweets in the DataFrame.
The code initializes an empty list tokens_list, and then loops over the indices of the DataFrame using range(len(cls)). For each index i, the process_tweet function is applied to the tweet text in the 'Tweet' column of the DataFrame at that index (cls['Tweet'].iloc[i]). The resulting list of cleaned and stemmed tokens for that tweet is then appended to the tokens_list variable.
After the loop completes, the tokens_list variable contains a list of all the cleaned and stemmed tokens for all the tweets in the DataFrame. The set function is then used to convert this list to a set of unique tokens.

STEP - 3: Feature Extraction/ Feature Engineering

Convert 'TimeOfDay' column into dummy variables

```
cls = pd.get_dummies(train_df, columns = ['TimeOfDay'])
```

This code creates dummy variables for the categorical variable 'TimeOfDay' in the train_df DataFrame using the pd.get_dummies() function.
The columns parameter specifies the column name for which to create the dummy variables, and pd.get_dummies() will create new columns for each unique value of 'TimeOfDay' in the DataFrame with a binary value indicating whether that value is present or not.
Since, the 'TimeOfDay' column has three unique values: from 0 to 23, pd.get_dummies(train_df, columns = ['TimeOfDay']) will create 24 new columns: 'TimeOfDay_0' to 'TimeOfDay_23', where each column will have a value of 1 if the corresponding value of 'TimeOfDay' was 0-23 respectively, and 0 otherwise. The resulting DataFrame cls will have the same columns as train_df, plus the new dummy variable columns.

Vectorizing the text (cleaned tweets)

```
from sklearn.feature_extraction.text import TfidfVectorizer
from imblearn.over_sampling import SMOTE
import scipy.sparse as sp

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(cls['Tweet'])
X_train_new_cols = sp.csr_matrix(cls[['text_sentiment', 'TimeOfDay_0', 'TimeOfDay_1', 'TimeOfDay_2', 'TimeOfDay_3', 'TimeOfDay_4', 'TimeOfDay_5', 'TimeOfDay_6', 'TimeOfDay_7', 'TimeOfDay_8', 'TimeOfDay_9', 'TimeOfDay_10', 'TimeOfDay_11', 'TimeOfDay_12', 'TimeOfDay_13', 'TimeOfDay_14', 'TimeOfDay_15', 'TimeOfDay_16', 'TimeOfDay_17', 'TimeOfDay_18', 'TimeOfDay_19', 'TimeOfDay_20', 'TimeOfDay_21', 'TimeOfDay_22', 'TimeOfDay_23']].values)
X = sp.hstack((X, X_train_new_cols))
y = np.array(cls['Label'])
vocabulary = vectorizer.vocabulary_
oversample = SMOTE()
X, y = oversample.fit_resample(X, y)

print(X.toarray().shape)
```

(5424, 7918)

The above code performs the following steps:

1. Imports the TfidfVectorizer from sklearn.feature_extraction.text and the SMOTE from imblearn.over_sampling packages.
2. It initializes the TfidfVectorizer object and fits it to the Tweet column of the DataFrame using the fit_transform() method. This step creates a sparse matrix of the tweet data with the Tf-idf scores.
3. It creates a sparse matrix from the additional columns in the DataFrame that contain information about the text sentiment and time of day of the tweet. These columns are converted to dummy variables using the pd.get_dummies() method before being converted to a sparse matrix using sp.csr_matrix(). The sparse matrices are then concatenated horizontally using sp.hstack() to create a single feature matrix X.
4. It creates a numpy array y containing the labels from the 'Label' column of the DataFrame.
5. Since the given data is an unbalanced dataset, we balance the whole dataset using SMOTE, which creates synthetic data for unbalanced labels.It creates an oversampled version of the feature matrix X and the label vector y using the SMOTE method from the imblearn package. The oversampled feature matrix X and label vector y are then returned.

We use X and y to train a prediction model using classification algorithms, where X is the feature matrix and y is the label vector, predicting for y.

STEP - 4: Training a Prediction model using Machine Learning

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3, random_state = 42, stratify = y)
```

This code chunk is splitting the oversampled data into training and testing sets using the train_test_split function from the sklearn.model_selection module. The input to the function includes the X and y arrays, representing the features and target variable respectively, along with the test_size parameter set to 0.3, indicating that 30% of the data will be used for testing.

Additionally, the random_state parameter is set to 42 to ensure that the same random data points are selected for each run of the code, and the stratify parameter is set to y to ensure that the class proportions are maintained in both the training and testing sets. The function returns four arrays: X_train, X_test, y_train, and y_test, representing the features and target variable for the training and testing sets. These arrays will be used in the subsequent steps of the machine learning pipeline.

```
from lazypredict.Supervised import LazyClassifier
clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)
models,predictions = clf.fit(np.array(X_train.todense()), np.array(X_test.todense()), np.array(y_train), np.array(y_test))

print(models)
```

```
100%|██████████| 29/29 [15:29<00:00, 32.04s/it]
                               Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
QuadraticDiscriminantAnalysis      0.91               0.91     0.91      0.91
SGDClassifier                      0.87               0.87     0.87      0.87
ExtraTreesClassifier               0.86               0.86     0.86      0.86
BernoulliNB                        0.85               0.85     0.85      0.85
RandomForestClassifier             0.85               0.85     0.85      0.85
Perceptron                         0.84               0.84     0.84      0.84
CalibratedClassifierCV             0.83               0.83     0.83      0.83
NearestCentroid                    0.82               0.82     0.82      0.82
PassiveAggressiveClassifier        0.82               0.82     0.82      0.82
LogisticRegression                 0.82               0.82     0.82      0.82
LGBMClassifier                     0.81               0.81     0.81      0.81
XGBClassifier                      0.81               0.81     0.81      0.81
LinearSVC                          0.81               0.81     0.81      0.81
BaggingClassifier                  0.80               0.80     0.80      0.80
RidgeClassifierCV                  0.80               0.80     0.80      0.79
KNeighborsClassifier               0.79               0.79     0.79      0.79
AdaBoostClassifier                 0.77               0.77     0.77      0.77
RidgeClassifier                    0.77               0.77     0.77      0.77
LinearDiscriminantAnalysis         0.77               0.77     0.77      0.76
DecisionTreeClassifier             0.76               0.76     0.76      0.76
GaussianNB                         0.75               0.75     0.75      0.74
ExtraTreeClassifier                0.74               0.74     0.74      0.74
NuSVC                              0.72               0.72     0.72      0.72
SVC                                0.69               0.69     0.69      0.68
LabelSpreading                     0.57               0.57     0.57      0.48
LabelPropagation                   0.57               0.57     0.57      0.48
DummyClassifier                    0.50               0.50     0.50      0.33
```

The LazyClassifier.fit method is then called with the training and testing data, and the resulting models and their predictions are stored in the variables "models" and "predictions", respectively.

Finally, the code prints the models and their associated metrics, including accuracy, balanced accuracy, F1-score, precision, recall, ROC AUC score, and training time. The models are sorted in descending order based on their F1-score, which is a measure of a model's accuracy and completeness.

Best Model = QuadraticDiscriminantAnalysis

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

mod = QuadraticDiscriminantAnalysis()
mod.fit(np.array(sp.csr_matrix(X_train).todense()), y_train)
y_pred = mod.predict(np.array(X_test.todense()))
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(accuracy,report)
```

```
0.9103194103194103
              precision    recall  f1-score   support

           0       0.87      0.97      0.92       814
           1       0.96      0.85      0.90       814

    accuracy                           0.91      1628
   macro avg       0.92      0.91      0.91      1628
weighted avg       0.92      0.91      0.91      1628
```

This is the accuracy and classification report on validation data with QuardraticDiscriminantAnalysis classifier model

EXTRA - Neural Networks Model

We can also use Neural Networks using a Multi Layer Perceptron classifier on the same features and labels.

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report

model_mlp = MLPClassifier(alpha=0.01, max_iter=500)
model_mlp.fit(X_train, y_train)
```

```
y_pred = model_mlp.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(accuracy,report)
```

```
0.8415233415233415              precision    recall  f1-score   support

           0       0.91      0.76      0.83       814
           1       0.79      0.92      0.85       814

    accuracy                           0.84      1628
   macro avg       0.85      0.84      0.84      1628
weighted avg       0.85      0.84      0.84      1628
```

This code trains a Multi-Layer Perceptron (MLP) classifier using the MLPClassifier class from the scikit-learn library. The alpha hyperparameter is set to 0.01 and the maximum number of iterations is set to 500. Then, the model is fitted on the training data using the fit() method. After that, the predict() method is used to generate predictions on the test data and the accuracy of the model is computed using the accuracy_score() function from the scikit-learn library. Finally, a classification report is generated using the classification_report() function from the scikit-learn library and printed to the console.

Overall, this code chunk evaluates the performance of an MLP classifier on the test data after training it on the training data.

Step - 5: Implement the same model on unseen Test Data

```
data_df = test_df
data_df['hashtags'] = data_df['Tweet'].apply(lambda x: re.findall(r'\#\w+', x))
data_df['hashtags'] = data_df['hashtags'].apply(lambda x: [re.sub(r'[^\w\s]','',h[1:]) for h in x])
data_df['hashtags_sentiment'] = data_df['hashtags'].apply(lambda x: [TextBlob(h).sentiment.polarity for h in x])
data_df['text_sentiment'] = data_df['Tweet'].apply(lambda x: TextBlob(x).sentiment.polarity)
test_df.Tweet = test_df.Tweet.apply(lambda x: process_tweet(x))
test_df['Tweet'] = test_df['Tweet'].apply(lambda x: ' '.join(x))
test_df = pd.get_dummies(test_df, columns = ['TimeOfDay'])
vectorizer = TfidfVectorizer(vocabulary = vocabulary)
X = vectorizer.fit_transform(test_df['Tweet'])
X_train_new_cols = sp.csr_matrix(test_df[['text_sentiment', 'TimeOfDay_0', 'TimeOfDay_1', 'TimeOfDay_2', 'TimeOfDay_3', 'TimeOfDay_4', 'TimeOfDay_5', 'TimeOfDay_6', 'TimeOfDay_7', 'TimeOfDay_8', 'TimeOfDay_9', 'TimeOfDay_10', 'TimeOfDay_11', 'TimeOfDay_12', 'TimeOfDay_13', 'TimeOfDay_14', 'TimeOfDay_15', 'TimeOfDay_16', 'TimeOfDay_17', 'TimeOfDay_18', 'TimeOfDay_19', 'TimeOfDay_20', 'TimeOfDay_21', 'TimeOfDay_22', 'TimeOfDay_23']].values)
X = sp.hstack((X, X_train_new_cols))
```

```
sub = pd.DataFrame()
sub['TweetID'] = test_df['TweetID']
y_pred = mod.predict(np.array(sp.csr_matrix(X).todense()))
sub['Label'] = y_pred
sub.to_csv('submissiono.csv',index=False)
```

The test data is also preprocessed using the same way how the train data was preprocessed and the features are extracted from the test data using the same vocabulary as in train data.
This code above is using the trained model to make predictions on the test data and create a submission file for the competition.
First, the test data is preprocessed by extracting hashtags, calculating their sentiment polarity, and calculating the overall sentiment polarity of each tweet using TextBlob. The process_tweet() function is applied to each tweet to tokenize it and remove noise.
Next, the get_dummies() function from pandas is used to create dummy variables for the categorical variable TimeOfDay.
Then, the same TfidfVectorizer object used to transform the training data is used to transform the preprocessed test data, but this time the vocabulary parameter is passed to the vectorizer to ensure consistency in the vocabulary between the training and test data. The dummy variables for TimeOfDay and the sentiment polarity features are added to the sparse matrix of transformed tweet data using hstack().
Finally, the predict() method of the trained mod model is used to predict the labels for the test data. The predictions are stored in a new dataframe called sub, which includes the TweetID and predicted label for each tweet. The to_csv() method is used to save this dataframe to a CSV file called submissiono.csv, which can be submitted to the competition.