In-Depth Analysis to Create NLP Prediction Model of Negative Event Types Likely to Occur for Medical Devices Via Multi-Label Classification

For this unsupervised learning project, multi-label classification in NLP was used to create a prediction model to predict what event type would likely occur for a medical device based on the recorded reasons of why the medical device was originally put under investigation. K-fold cross validation was utilized to ensure the performance of the prediction model. Below documents how in-depth analysis was used to create the prediction model via multi-label classification with NLP performed on the textual reason column from the dataset in the df_final.csv and with the type column from the same dataset being denoted as the target variable.

Before the prediction model was created, the data from the df_final dataset underwent the k-fold cross validation procedure as well as needed its text data and target variable converted into features and formats respectively that can be used in the prediction model. Basically, cross validation is a resampling technique used to evaluate machine learning models on a limited data set, and k-fold cross validation is the most common cross validation procedure. The k-fold cross validation procedure has a single parameter called k that represents the number of sub-groups that a given dataset is to be split into. In this project, the k-fold cross validation method was used over the simple train/test split due to the method generally having a less biased or less optimistic estimate of the model's skill. This method works by randomly dividing the set of the data's observations into approximately equal-sized k groups, or folds. The first fold is treated as the validation (test) set while the method is fitted on the remaining $k - 1$ folds (training set). To prepare the df_final dataset for the k-fold cross validation procedure, the clean_reason column which contains the cleaned text data of the reason column and the type column were copied from the df_final dataset to the sub-dataframe denoted as data. The clean_reason column and the type column in the data sub-dataframe were renamed "text" and "class" respectively. To have the sub-dataframe being randomized further during the k-fold cross validation procedure, the index of the data sub-dataset was randomly permutated.

```
#Preparing data for k-fold cross-validation
data = df_final[['clean_reason', 'type']].copy()
data = data.rename(columns={'clean_reason': 'text', 'type': 'class'}) #rename 
the columns in data
data = data.reindex(np.random.permutation(data.index))
```

Next, the features of the text in the text column were extracted (reducing the mass of unstructured data into some uniform set of features that the algorithm can learn from). For text classification, word count frequencies were used on the corpus (the union collection of texts) in correspondence to each class of the event types. The CountVectorizer function of Python is perfect for the feature extraction as it converts a collection of text documents into a matrix of token counts. In order to further improve the results of the prediction model, more features were extracted from the text data in the text column of the data sub-dataset. This was done by utilizing n-gram counts instead of simply relying on word counts which uses bag-of-words features (All of the words are tossed into a "bag" and counted without regard to any contextual meaning that could be associated with the ordering of words). An n-gram is a sequence of ordered words of length n. For example, in the sentence, “Please, turn in your homework,” a 2-gram (or bigram) sequence will be “please turn,” “turn your,” or ”your homework” while a 3-gram (or trigram) sequence will be “please turn your” or “turn your homework”. The CountVectorizer function can include any order of n-grams by giving it a range. For the data sub-dataset, bigrams seemed to give the k-fold cross validation algorithm the best boost in accuracy. Even if trigrams gave a bit more accuracy to the algorithm, it incurred a longer computation time which made it not worth the infinitesimal increase. With word count frequencies extracted as features, the next step would be training a classifier and classifying the corpus. A naïve Bayes classifier was used to do so as it assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature, given the class variable. In other words, each feature (in this case word counts) is independent from every other feature, and each one contributes to the probability that an example belongs to a particular class. The MultinomialNB function in Python was used as the classifier and trained by passing in the feature vector and the target vector (the classes that each example belongs to). The indices of each vector must be aligned, but Pandas automatically does that. The Pipeline function from the Scikit Learn library was used to merge the feature extraction and classification into one operation. In other words, the Pipeline function pipelines a series of steps into one object which can be trained and then used to make predictions.

```
#Create a pipeline using a naïve Bayes classifier
pipeline = Pipeline([
  ('vectorizer',  CountVectorizer(ngram_range=(1, 2))), #extract more
features from the text by implementing bigram (2-grams) counts
  ('classifier',  MultinomialNB())])
```

With the features extracted from the data sub-dataset and classified, the k-fold cross validation was ready to be utilized. Scikit Learn's KFold was used to perform the k-fold cross validation by generating k pairs of bitmasks – vectors of booleans which were used to select out randomized portions of the dataset for the train and test sets. The shuffle option in the KFold function was set to true in order to shuffle the data before splitting it into k parts. After the data sub-dataset was shuffled, it was split into k parts. Then, one of the parts was held out while the others were combined. Finally, the k-fold cross validation trained on the combined parts and then validated against the held-out portion. The process was repeated k times (number of folds), holding out a different portion each time. In this project, the value of k was fixed to 10 as it is a value that has been found through experimentation to generally result in a model skill estimate with low bias and a modest variance.

```
#Performing k-fold cross-validation
k_fold = KFold(n_splits=10, shuffle=False, random_state=None) #set 10 folds
to the cross-validation
scores = []
for train_indices, test_indices in k_fold.split(data):
  train_text = np.asarray(data.iloc[train_indices]['text'])
  train_y    = np.asarray(data.iloc[train_indices]['class'])
  test_text  = np.asarray(data.iloc[test_indices]['text'])
  test_y     = np.asarray(data.iloc[test_indices]['class'])

  pipeline.fit(train_text, train_y)
  score = pipeline.score(test_text, test_y)
  scores.append(score)

score = sum(scores) / len(scores)
```

Scikit Learn models provide a score method that calculates the mean accuracies of each fold which are then averaged together for a mean accuracy of the entire k-fold cross validation set.

```
#Check the mean accuracy score of the k-fold cross-validation
score
```

```
0.94851247432078
```

The mean accuracy of the k-fold cross validation in this project was calculated to be about 95%. This means that the train and test sets created from the k-fold cross validation are about 95% accurate overall. This is a pretty high accuracy which is needed for the NLP prediction model to be created in this project. The train and test sets of the text data were converted into TF-IDF (Term Frequency — Inverse Document Frequency) features.

```
#Using TF-IDF to extract features from the cleaned version of the reason data
tfidf_vectorizer = TfidfVectorizer(max_df=0.8, max_features=10000) # set 10,0
00 most frequent words in the data as features

#Create TF-IDF features
xtrain_tfidf = tfidf_vectorizer.fit_transform(train_text)
xval_tfidf = tfidf_vectorizer.transform(test_text)
```

This means that the words in the text were quantified based on the computed weight of each word which signifies the importance of the word in the corpus. The train and test sets of the target variable were also formatted to be one-hot encoded by sklearn’s MultiLabelBinarizer function which transforms a list of sets or tuples into the supported multilabel format - a (samples x classes) binary matrix indicating the presence of a class label. This was done by converting the train and test sets of the target variable into dataframes before splitting their values into separate lists.

```
#Convert train_y and test_y into datasets
df_train_y=pd.DataFrame(train_y, columns=['new_type'])
df_test_y=pd.DataFrame(test_y, columns=['new_type'])

#Converting values in df_train_y and df_test_y into lists with ',' replacing
'/'
df_train_y['new_type'] = df_train_y['new_type'].str.split(' / ')
df_test_y['new_type'] = df_test_y['new_type'].str.split(' / ')
```

Then, they were fitted and transformed by the MultiLabelBinarizer function into the binary matrix format that can be used in the prediction model for classifying the text data.

```
#One hot encode the target variable, i.e., type by using sklearn’s MultiLabel
Binarizer( )
multilabel_binarizer = MultiLabelBinarizer()
multilabel_binarizer.fit(df_train_y['new_type'])
multilabel_binarizer.fit(df_test_y['new_type'])
```

```
MultiLabelBinarizer(classes=None, sparse_output=False)
```

```
# transform target variable
ytrain = multilabel_binarizer.transform(df_train_y['new_type'])
yval = multilabel_binarizer.transform(df_test_y['new_type'])
```

With the features of the text data (clean_reason column from the df_final dataset) extracted, and the target variable (type column from the df_final dataset) formatted, the NLP prediction model is ready to be created. Building a model for every one-hot encoded target variable would take a considerable amount of time. Thus, a Logistic Regression model would be built as it is quick to train on limited computational power with sklearn’s OneVsRestClassifier class used to solve the model’s problem as a Binary Relevance or a one-vs-all problem.

```
#Use sk-learn’s OneVsRestClassifier class to solve this problem as a Binary R
elevance or one-vs-all problem
lr = LogisticRegression()
clf = OneVsRestClassifier(lr)
```

The model was fitted on the train sets of the TF-IDF features and the formatted target variable.

```
#Fit model on train data
clf.fit(xtrain_tfidf, ytrain)
```

After the model was trained, it was used to make predictions with the validation (or test) TF-IDF features set.

```
#Make predictions for validation set
y_pred = clf.predict(xval_tfidf)
```

Displaying a sample of the predictions only shows it as a binary one-dimensional array (or the one-hot encoded form of the event types tags).

```
#Check out a sample from these predictions
y_pred[3]
```

```
array([0, 1, 0])
```

Fortunately, sklearns’ inverse_transform function and the MultiLabelBinarizer object helped to convert the predicted arrays into event type tags.

```
#Convert the predicted arrays into event type tags
multilabel_binarizer.inverse_transform(y_pred)[3]
```

```
('Recall',)
```

To evaluate the model's overall performance, all of the predictions and the entire target variable of the validation set would need to be taken into consideration. The F1 score, also known as balanced F-score or F-measure, was used to evaluate the overall performance of the prediction model. The F1 score is interpreted as a weighted average of the precision and recall where 1 is its best score, and 0 is its worst score. For the F1 score, the relative contribution of precision and recall are equal. The formula for the F1 score is:

$$F1 = 2 \times (\text{precision} \times \text{recall}) / (\text{precision} + \text{recall})$$

In the multi-class and the multi-label case, the F1 score is the average of each class with weighting depending on the average parameter.

```
# evaluate performance
f1_score(yval, y_pred, average="micro")
```

```
0.9596819988642816
```

The F1 score was calculated to be about 0.96. This means that the prediction model that was created in this project has nearly perfect precision and recall. Since the prediction model has a nearly perfect F1 score, an inference function was created to take any text describing the reasons for a medical device being put under investigation as input and to give the predicted outcome of the negative event type(s) (Field Safety Notice, Recall, and/or Safety Alert) that would take place for the medical device. First, the type column in the df_final dataset was split to convert the values in the column into individual lists which were stored into a new column called new_type.

```
#Create new_type column in df_final dataset
df_final['new_type'] = df_final['type'].str.split(' / ')
```

Then the text cleaning functions that were created in Capstone Project 1 Dataset - Final-Edited.ipynb were loaded.

```
# function for text cleaning
def clean_text(text):
    # remove everything except alphabets
    text = re.sub("[^a-zA-Z]"," ",text)
    # remove whitespaces
    text = ' '.join(text.split())
    # convert text to lowercase
    text = text.lower()

    return text

# function for stemming words
def stem_text(text):
    ps = PorterStemmer()
    token_words=word_tokenize(str(text))
    token_words
    stem_text=[]
    for word in token_words:
        stem_text.append(ps.stem(word))
        stem_text.append(" ")
    return "".join(stem_text)

# function to remove stopwords
def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    no_stopword_text = [w for w in str(text).split() if not w in stop_words]
    return ' '.join(no_stopword_text)
```

With the formatted type (new_type) column from the df_final dataset, the loaded text cleaning functions, and the prediction model of this project, the infer_event function was created.

```
#Create inference function
def infer_event(q):
    q = clean_text(q)
    q = stem_text(q)
    q = remove_stopwords(q)
    q_vec = tfidf_vectorizer.transform([q])
    q_pred = clf.predict(q_vec)
    return multilabel_binarizer.inverse_transform(q_pred)
```

The infer_event function first cleans the text by removing non-alphabetical texts and white spaces, converting the text to lowercase, stemming the text, and removing stop words before extracting features from the text via the TfidfVectorizer function in Python to make predictions which are returned as negative medical event type tags. The Python code below displays the results of the infer_event function worked on ten samples from the corpus under the reason column in the df_final dataset.

```
for i in range(10):
  k = df_final['reason'].sample(1).index[0]
  print("Medical Device: ", df_final['device_name'][k], "\nPredicted Event
Type: ", infer_event(df_final['reason'][k])), print("Actual Event Type:
",df_final['new_type'][k], "\n")
```

```
Medical Device:  ARCHITECT SYSTEM - PROGESTERONE ASSAY
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']

Medical Device:  Single Shot Epidural Anesthesia Kit, Internal Jugular Pun
cture Kit with Blue FlexTip(R) Catheter, Pediatric Jugular Puncture Kit, A
rterial Line Kit, Vessel Catheterization kit, Central Venous Catheterizati
on kit, Jugular Puncture Ks. jne. ks.ilm.,
Predicted Event Type:  [('Recall',)]
```

```
Actual Event Type:  ['Recall']

Medical Device:  Device Recall  Cytosponge Cell Collection Device
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']

Medical Device:  Device Recall  Howell D.A.S.H. Extraction Balloon
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']

Medical Device:  CHECKCELLS (POOLED CELLS)
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']

Medical Device:  RESONATE, VIGILANT X4, PERCIVA, and MOMENTUM CRT-D/ICD
Predicted Event Type:  [('Safety Alert',)]
Actual Event Type:  ['Safety Alert']

Medical Device:  LOW PROFILE ABUTMENTS  INTERNAL AND EXTERNAL CONNECTION
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']

Medical Device:  AVANTA FLUID INJECTION SYSTEM - MAIN UNIT WITH PEDESTAL
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']

Medical Device:  WASHER AND AUTOMATIC ENDOSCOPE REPAIRER AER - registered
in Anvisa under the number 80145900728 - Code 20301 - Serial numbers / Lot
s: EP1151741; EP1151742; EP1151744; EP1151747; EP1151715; EP1151568; EP115
1390; 4024730; 3024357; EP1151459; EP115119; EP115910; EP1151116.
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Safety Alert']

Medical Device:  Device Recall  Giraffe Infant Warmers
Predicted Event Type:  [('Recall',)]
Actual Event Type:  ['Recall']
```

The NLP prediction model especially in the infer_event function looks very serviceable. However, this model was only worked on English medical text. Thus, it would likely have limitations on medical text written in different languages as different languages have different grammatical structures. Plus, as countries have different medical policies and follow different medical systems, there might be different medical terminology and lingo between individual countries. Thus, the prediction model created in this project would most successfully work on English-language medical text that follow the same medical policies and systems for negative event type classifications concerning medical devices.