In [1]:

```
from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, svm
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn import decomposition, ensemble

import pandas, xgboost, numpy, textblob, string
from keras.preprocessing import text, sequence
from keras import layers, models, optimizers
from nltk import word_tokenize
from nltk.corpus import stopwords
import sklearn
#import sklearn_crfsuite
#from sklearn_crfsuite import scorers
#from sklearn_crfsuite import metrics
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics import accuracy_score
from sklearn import metrics
```

```
C:\Users\acharya\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of
the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it wi
ll be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [2]:

```
import nltk
nltk.download('stopwords')
stopwords = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\acharya\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [3]:

```
# load the dataset # merge_from_ofoct.txt contains combined labels and text from all training papers
data = open('labeled_sentences (1).txt').read()
labels, texts = [], []
for i, line in enumerate(data.split("\n")):
    content = line.split()
    #print(content)
    labels.append(content[0])
    filtered_sentence = [w.lower() for w in content[1:] if not w in stopwords]
    texts.append(filtered_sentence)

# create a dataframe using texts and lables
trainDF = pandas.DataFrame()
trainDF['text'] = texts
trainDF['label'] = labels
print(texts[0])
```

```
['minimum', 'description', 'length', 'principle', 'online', 'sequence', 'estimation/prediction'
, 'proper', 'learning', 'setup', 'studied']
```

In [4]:

```
# split the dataset into training and validation datasets
train_x, valid_x, train_y, valid_y = model_selection.train_test_split(trainDF['text'], trainDF['label'])



# label encode the target variable
encoder = preprocessing.LabelEncoder()
train_y = encoder.fit_transform(train_y)
valid_y = encoder.fit_transform(valid_y)
```

In [5]:

```
tempp = []

for item in train_x:
    tempp.append(" ".join(item))
#print(len(train_x))

tempp1 =[]
for item1 in valid_x:
    tempp1.append(" ".join(item1))

#print(tempp1)

temp =[]
temp_len=0
for item2 in texts:
    temp.append(" ".join(item2))
    temp_len = temp_len+len(texts)
print(len(temp))
print(temp_len)
print(type(temp))
```

```
18627
346965129
<class 'list'>
```

In [6]:

```
# create a count vectorizer object
count_vect = CountVectorizer(analyzer='word', token_pattern=r'\w{1,}')
count_vect.fit(temp)

# transform the training and validation data using count vectorizer object
xtrain_count =  count_vect.transform(tempp)
xvalid_count =  count_vect.transform(tempp1)
```

In [7]:

```
# word level tf-idf
tfidf_vect = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', max_features=5000)
tfidf_vect.fit(temp)
xtrain_tfidf =  tfidf_vect.transform(tempp)
xvalid_tfidf =  tfidf_vect.transform(tempp1)

# ngram level tf-idf
tfidf_vect_ngram = TfidfVectorizer(analyzer='word', token_pattern=r'\w{1,}', ngram_range=(2,3), max_features
=5000)
tfidf_vect_ngram.fit(temp)
xtrain_tfidf_ngram =  tfidf_vect_ngram.transform(tempp)
xvalid_tfidf_ngram =  tfidf_vect_ngram.transform(tempp1)

# characters level tf-idf
tfidf_vect_ngram_chars = TfidfVectorizer(analyzer='char', token_pattern=r'\w{1,}', ngram_range=(2,3), max_fe
atures=5000)
tfidf_vect_ngram_chars.fit(temp)
xtrain_tfidf_ngram_chars =  tfidf_vect_ngram_chars.transform(tempp)
xvalid_tfidf_ngram_chars =  tfidf_vect_ngram_chars.transform(tempp1)
```

In [8]:

```
def train_model(classifier, feature_vector_train, label, feature_vector_valid, is_neural_net=False):
    # fit the training dataset on the classifier
    #std_clf = make_pipeline(StandardScaler(with_mean=False), TruncatedSVD(100), MultinominalNB())
    #std_clf.fit(feature_vector_train, label)
    classifier.fit(feature_vector_train, label)

    # predict the labels on validation dataset
    #predictions = classifier.predict(feature_vector_valid)
    predictions = classifier.predict(feature_vector_valid)
    if is_neural_net:
        predictions = predictions.argmax(axis=-1)

    return metrics.accuracy_score(predictions, valid_y)
```

In [9]:

```
# Naive Bayes on Count Vectors
accuracy = train_model(naive_bayes.MultinomialNB(), xtrain_count, train_y, xvalid_count)
print ("NB, Count Vectors: ", accuracy)
# Naive Bayes on Word Level TF IDF Vectors
accuracy = train_model(naive_bayes.MultinomialNB(), xtrain_tfidf, train_y, xvalid_tfidf)
print ("NB, WordLevel TF-IDF: ", accuracy)
# Naive Bayes on Ngram Level TF IDF Vectors
accuracy = train_model(naive_bayes.MultinomialNB(), xtrain_tfidf_ngram, train_y, xvalid_tfidf_ngram)
print ("NB, N-Gram Vectors: ", accuracy)
# Naive Bayes on Character Level TF IDF Vectors
accuracy = train_model(naive_bayes.MultinomialNB(), xtrain_tfidf_ngram_chars, train_y, xvalid_tfidf_ngram_ch
ars)
print ("NB, CharLevel Vectors: ", accuracy)
```

```
NB, Count Vectors:  0.847970796650204
NB, WordLevel TF-IDF:  0.8400257676615848
NB, N-Gram Vectors:  0.8048099634958128
NB, CharLevel Vectors:  0.7904230191110156
```

In [11]:

```
# SVM on Ngram Level TF IDF Vectors
accuracy = train_model(svm.SVC(kernel= 'rbf',random_state=0, gamma=1, C=1), xtrain_tfidf_ngram, train_y, xva
lid_tfidf_ngram)
print ("SVM, N-Gram Vectors: ", accuracy)
```

```
SVM, N-Gram Vectors:  0.7990122396392527
```

In [10]:

```
# RF on Count Vectors
accuracy = train_model(ensemble.RandomForestClassifier(), xtrain_count, train_y, xvalid_count)
print ("RF, Count Vectors: ", accuracy)

# RF on Word Level TF IDF Vectors
accuracy = train_model(ensemble.RandomForestClassifier(), xtrain_tfidf, train_y, xvalid_tfidf)
print ("RF, WordLevel TF-IDF: ", accuracy)
```

```
RF, Count Vectors:  0.8707322310500322
RF, WordLevel TF-IDF:  0.8750268413141508
```

In [12]:

```
# load the pre-trained word-embedding vectors
embeddings_index = {}
for i, line in enumerate(open('crawl-300d-2M.vec','r', encoding='utf-8')):

    values = line.split()
    embeddings_index[values[0]] = numpy.asarray(values[1:], dtype='float32')

# create a tokenizer
token = text.Tokenizer()
token.fit_on_texts(temp)
word_index = token.word_index

# convert text to sequence of tokens and pad them to ensure equal length vectors
train_seq_x = sequence.pad_sequences(token.texts_to_sequences(tempp), maxlen=70)
valid_seq_x = sequence.pad_sequences(token.texts_to_sequences(tempp1), maxlen=70)

# create token-embedding mapping
embedding_matrix = numpy.zeros((len(word_index) + 1, 300))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

In [21]:

```
from keras.utils import to_categorical

y_binary = to_categorical(train_y)
def create_cnn():
    # Add an Input Layer
    input_layer = layers.Input((70, ))

    # Add the word embedding Layer
    embedding_layer = layers.Embedding(len(word_index) + 1, 300, weights=[embedding_matrix], trainable=False
)(input_layer)
    embedding_layer = layers.SpatialDropout1D(0.3)(embedding_layer)

    # Add the convolutional Layer
    conv_layer = layers.Convolution1D(100, 3, activation="relu")(embedding_layer)

    # Add the pooling Layer
    pooling_layer = layers.GlobalMaxPool1D()(conv_layer)




    # Add the output Layers
    output_layer1 = layers.Dense(50, activation="relu")(pooling_layer)
    output_layer1 = layers.Dropout(0.25)(output_layer1)
    output_layer2 = layers.Dense(5, activation="sigmoid")(output_layer1)

    # Compile the model
    model = models.Model(inputs=input_layer, outputs=output_layer2)
    model.compile(optimizer=optimizers.Adam(), loss='categorical_crossentropy')

    return model

classifier = create_cnn()
accuracy = train_model(classifier, train_seq_x, y_binary, valid_seq_x, is_neural_net=True)
print ("CNN, Word Embeddings",  accuracy)
```

```
Epoch 1/1
13970/13970 [==============================] - 14s 999us/step - loss: 0.6290
CNN, Word Embeddings 0.8460382220313507
```

In [24]:

```
def create_rnn_lstm():
    # Add an Input Layer
    input_layer = layers.Input((70, ))

    # Add the word embedding Layer
    embedding_layer = layers.Embedding(len(word_index) + 1, 300, weights=[embedding_matrix], trainable=False
)(input_layer)
    embedding_layer = layers.SpatialDropout1D(0.3)(embedding_layer)

    # Add the LSTM Layer
    lstm_layer = layers.LSTM(100)(embedding_layer)

    # Add the output Layers
    output_layer1 = layers.Dense(50, activation="relu")(lstm_layer)
    output_layer1 = layers.Dropout(0.25)(output_layer1)
    output_layer2 = layers.Dense(5, activation="sigmoid")(output_layer1)

    # Compile the model
    model = models.Model(inputs=input_layer, outputs=output_layer2)
    model.compile(optimizer=optimizers.Adam(), loss='categorical_crossentropy')

    return model

classifier = create_rnn_lstm()
accuracy = train_model(classifier, train_seq_x, y_binary, valid_seq_x, is_neural_net=True)
print ("RNN-LSTM, Word Embeddings",  accuracy)
```

```
Epoch 1/1
13970/13970 [==============================] - 29s 2ms/step - loss: 0.6638
RNN-LSTM, Word Embeddings 0.8045952329826068
```

In [26]:

```
def create_bidirectional_rnn():
    # Add an Input Layer
    input_layer = layers.Input((70, ))

    # Add the word embedding Layer
    embedding_layer = layers.Embedding(len(word_index) + 1, 300, weights=[embedding_matrix], trainable=False
)(input_layer)
    embedding_layer = layers.SpatialDropout1D(0.3)(embedding_layer)

    # Add the LSTM Layer
    lstm_layer = layers.Bidirectional(layers.GRU(100))(embedding_layer)

    # Add the output Layers
    output_layer1 = layers.Dense(50, activation="relu")(lstm_layer)
    output_layer1 = layers.Dropout(0.25)(output_layer1)
    output_layer2 = layers.Dense(5, activation="sigmoid")(output_layer1)

    # Compile the model
    model = models.Model(inputs=input_layer, outputs=output_layer2)
    model.compile(optimizer=optimizers.Adam(), loss='binary_crossentropy')

    return model

classifier = create_bidirectional_rnn()
accuracy = train_model(classifier, train_seq_x, y_binary, valid_seq_x, is_neural_net=True)
print ("RNN-Bidirectional, Word Embeddings",  accuracy)
```

```
Epoch 1/1
13970/13970 [==============================] - 43s 3ms/step - loss: 0.2321
RNN-Bidirectional, Word Embeddings 0.8443203779257032
```

In [27]:

```
def create_rcnn():
    # Add an Input Layer
    input_layer = layers.Input((70, ))

    # Add the word embedding Layer
    embedding_layer = layers.Embedding(len(word_index) + 1, 300, weights=[embedding_matrix], trainable=False
)(input_layer)
    embedding_layer = layers.SpatialDropout1D(0.3)(embedding_layer)

    # Add the recurrent layer
    rnn_layer = layers.Bidirectional(layers.GRU(50, return_sequences=True))(embedding_layer)

    # Add the convolutional Layer
    conv_layer = layers.Convolution1D(100, 3, activation="relu")(embedding_layer)

    # Add the pooling Layer
    pooling_layer = layers.GlobalMaxPool1D()(conv_layer)

    # Add the output Layers
    output_layer1 = layers.Dense(50, activation="relu")(pooling_layer)
    output_layer1 = layers.Dropout(0.25)(output_layer1)
    output_layer2 = layers.Dense(5, activation="sigmoid")(output_layer1)

    # Compile the model
    model = models.Model(inputs=input_layer, outputs=output_layer2)
    model.compile(optimizer=optimizers.Adam(), loss='binary_crossentropy')

    return model

classifier = create_rcnn()
accuracy = train_model(classifier, train_seq_x, y_binary, valid_seq_x, is_neural_net=True)
print ("CNN, Word Embeddings",  accuracy)
```

```
Epoch 1/1
13970/13970 [==============================] - 15s 1ms/step - loss: 0.2251
CNN, Word Embeddings 0.8501181017822632
```