**Code Review**

**Pandas** 🡪

***used to preprocess the data***: First we loaded the dataset.

***Dataset***: it contains 4 columns 🡪 Sentence #, Pos, Word, Tag.

: it contains **1048575** record

: it contains **47959** sentences  
: it contains **35178** words

We knew all of that via pandas and its powerful preprocessing features.

**Keras** 🡪

***Pad sequencing*:** is used to convert the data into vectors to make the prediction efficient for the model and more easy.

X = pad\_sequences(maxlen=max\_length, sequences=X, padding="post", value=n\_words - 1)

We took the words tuple ***“Words column from the dataset”*** and convert it to vectors.

**To\_categorical**: used to convert a vector to a matrix of zeros and ones.

y = [to\_categorical(i, num\_classes=n\_tags) for i in y]

**Building the Bidirectional LSTM model**: ***Embedding*** to build the first layer.

model = Embedding(input\_dim=n\_words, output\_dim=50, input\_length=max\_length)(input)

***Dropout*** which is used to drop a percentage of the training data to avoid overfitting.

model = Dropout(0.1)(model)

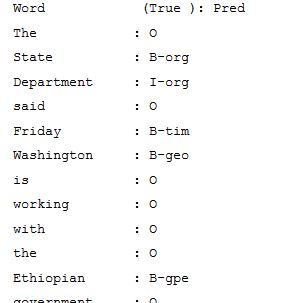
***Bidirectional*** this is a wrapper for the RNN and makes a bidirectional path between the nodes, this allows the network to have both forward and backward information about the sequence at every step, this can work as a memory for the network to help it to predict efficiently.

***model = Bidirectional(LSTM(units=100, return\_sequences=True, recurrent\_dropout=0.1))(model)***

***TimeDistributed*** You can then use `TimeDistributed` to apply a `Dense` layer to each of the 10 timesteps, independently

out = TimeDistributed(Dense(n\_tags, activation="softmax"))(model)

**Predictions**

****

**i = 2318**

**p = model.predict(np.array([X\_te[i]]))**

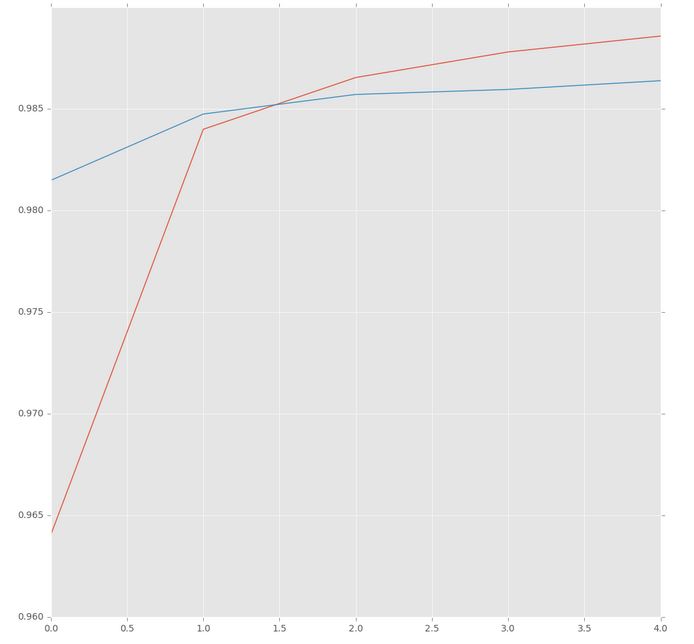
**p = np.argmax(p, axis=-1)**

**print("{:15} ({:5}): {}".format("Word", "True", "Pred"))**

**for w, pred in zip(X\_te[i], p[0]):**

**print("{:15}: {}".format(words\_col[w], tags\_col[pred]))**

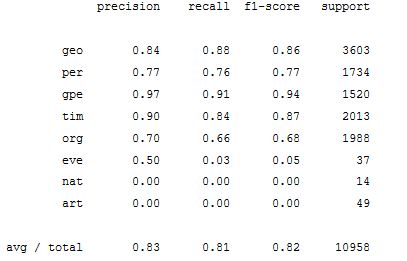
**Model Accuracy**

****

**Accuracy with red**

**Val\_acc with blue**

***Evaluation***

******

***F1 score = 67.1%***

******

***Precision is calculated via***

******

***Recall***

******

***F1-Score***

Supervised Documentation

***MultinomialNB🡪***

The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts

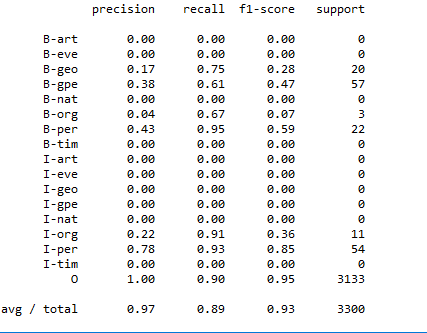
***Score🡪***

***Using 1000 samples , with 100 random state 🡪 0.8918181818181818***

***Train size :*** ***6700***

***Test size : 3300***

***Summary***

******