# Contract Clasification

## Team Group G (CS985)


Elliot Richardson (202079108)

Keshan Sharp (202068088)

Ben Litherland (202059817)

Ana Hernandez (202058338)


# Introduction

This is a multi label classification problem for which, given the data for a contract document, it's categories can be predicted. In this example, there are 9 possible categories a document can belong to. With the use of NLP, we were able to efficiently manipulate the text input data, which was the sole input for most of the predictive models.

A simple SVM and a dense, deep neural network were developed before exploring architectures typically better suited to this problem of text classification, like LSTM[1] and GRU units. The main issues encountered with this dataset were feature engineering with the natural language sections of 'title','description', and 'awarding authority'. Concatenating these features simplified the processing of these data, and yielding impressive results.

# Main Findings and Discussion

The Naive Bayes classifier performed surprisingly well for a shallow learner making use of text data only, scoring 0.92961 on the test set - almost outperforming the final NN (Model B) with a score of 0.93790. The worst performing model was the stardard feed forward NN with a Macro-F1 score of 0.104. The state of the art models both reported Macro-F1 scores in excess of 0.93, demonstrating large gains over the standard NN architecture.

## Data processing

The EDA and pre-processing of the dataset was completed in the attached notebook. It was seen that there is a high frequency for a handful of the categories and many instances of only 1 or two labels. The are 9! combinations of the possible sectors as labels but it's clear in the real world some combinations are more common. There is no clear method for identifying outliers from the text data, thus no augmentation or outlier pruning was undertaken. One interesting change was the conversion of 'publication_date' to a categorical variable where each date was replaced with the month of that date. This yielded slightly better performance for Model B.

The three string attributes in the dataset ("title", "description","awarding authority") were concatenated to form a single text input. This was subsequently vectorized to enable NLP. This was carried out with Keras' Tokenizer class. Tokenisation removes all punctuation, converting all remaining text into lowerclass. With this, it creates a dictionary with the unique, significant words. The generated vector represent the words

## Method

### Baseline Naive Bayes Classifier

When looking for a simple classifier we adapted a solution to multi-label classification demonstrated on IMDB movie review sentiment analysis [2,3]. This set a high bar for our models but it was good to see how simple systems could yield extremely accurate results with minimal processing and CPU usage. A series of 'One vs All' models compile the preditions for each possible label and the results are concatenated together.

### Neural Network

We employed the same grid search approach that was applied to the first problem, with a NN builder and optimiser to search through the pertinent hyperparameters. Even when the number of nodes ranged between 10-200 and the number of hidden layers between 1-9 no satisfactory results were achieved. Even with this it still only produced 2 unique predictions and scored poorly on kaggle.

### Model A

**Layer 1**:

We create an embedding layer by specifying the input size of the mapping, output size, and input length (amount of words per sequence). We add 1 to the vocabulary size because 0 is a reserved value that we use for padding. If the entry has less words than the stated max length, it will fill in the vector with 0's up to max length. However, if its longer, it will only use the first 400 values.

**Layer 2**:

Adding the LSTM layer to the model, we need to set the number of units. This number corresponds to the dimensionality of the output space and thus also the dimensionality of the cell state, the hidden state, and the NN gates.

To complete the classifier, we can then add two dense layers, a sigmoid output layer, producing a range between 0 and 1, corresponding to 'no' or 'yes' to each category. The dropout layer randomly sets a proportion of its input units to zero. This is added to prevent overfitting as it reduces the NN's dependence on certain features.

### Model B

This model is an extension of Model A's architecture, altered to use GRU units instead, and it's output concatenated with those of a feed-forward NN before being fed through further Dense layers with dropout. This allowed for categorical data to be used for prediction alongside the text data and a plot of this network can be seen below.

In a similar multilabel, multiclass problem[5] the binary crossentropy and adam optimiser were used, thus they were chosen again as default parameters during experimentation, remaining the best choice.

## Training schedule

The first model carried out was the baseline Naive Bayes classifier, with little parameter tuning this model performed incredibly well in its first run. The same preprocessing carried out for the baseline was utilised for our more complex models. For Models A and B, we looked at many examples of similar problems and selected the recommended parameters for "max length sequence" and "embedding dimensions", also for the number of layers and their units

## Results and Discussion

The results for our validation sets were very high for the NB, A and B models. Therefore we thought we could be overfitting the models to the training data. However, when submitted to Kaggle, we ruled out this possibilty due to their high performance.

Below are the performance metrics for the finalised models on our validation sets and test set.

| | Naive bayes | Tuned 3 Layer NN | Model A | Model B |
|---|---|---|---|---|
| Loss | N/A | 0.279 | 0.077 | 0.078 |
| Accuracy | 0.991 | 0.565 | 0.933 | 0.931 |
| Precision | 0.991 | 0.601 | 0.962 | 0.958 |
| Recall | 0.932 | 0.545 | 0.955 | 0.956 |
| F1 Train | 0.960 | 0.572 | 0.922 | 0.921 |
| Kaggle Score | 0.937 | 0.104 | 0.933 | 0.937 |

## Summary and Recommendation

The **Naive Bayes Classifier** was trained in under 2 minutes and little parameter tuning was needed, which makes this very adaptable to different input text data.

**Model A** was trained in arond 30-40 minutes and the number of layers and nodes was to be explored and inspired with similar text processing models. Some parameters must be tuned according to the input data, like the max length of word vectors, embedding dimensions etc.. for higher efficiency.

**Model B** was trained in around 30-40 minutes but it trained with text and categorical input data. Because we only had a few informative categorical attributes, this didn't make a huge difference in terms of F1-Score. Similar parameter tuning to model A will be needed for adapting to different datasets. This model scored the highest F1-Score.

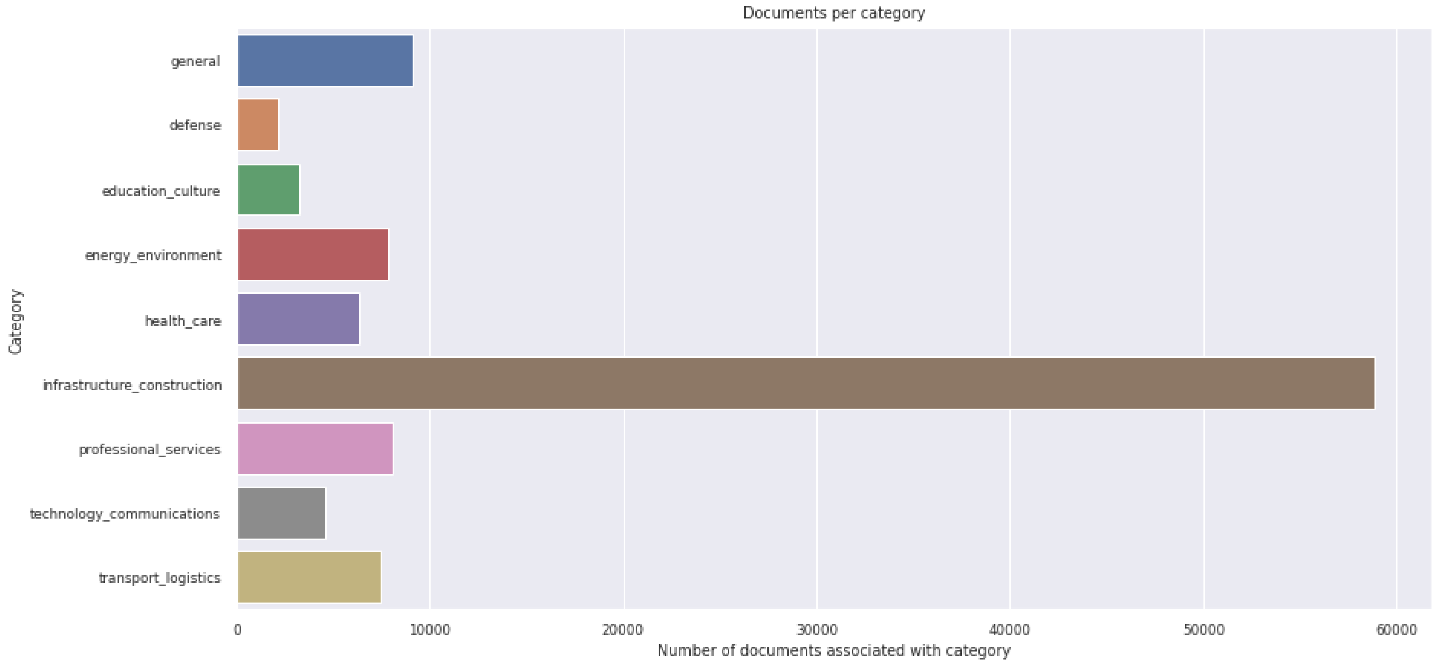
In conclusion, the performance of 3 out of 4 of the models are quite similar. However, the training time and cost for each one varied. In terms of simplicity we would recommend the baseline classifier as it's highly adaptable to any length, type, and language of the input text data. However, model B uses categorical too, and could be easily adapted to taking in numerical values too. Therefore, if the dataset had more categorical or numerical properties of the contract documents, model B may make better use of all the data and would outperform the baseline classifier. Therefore, the most adequate model choice will depend on the descriptors of the documents the company will be given, and their type.

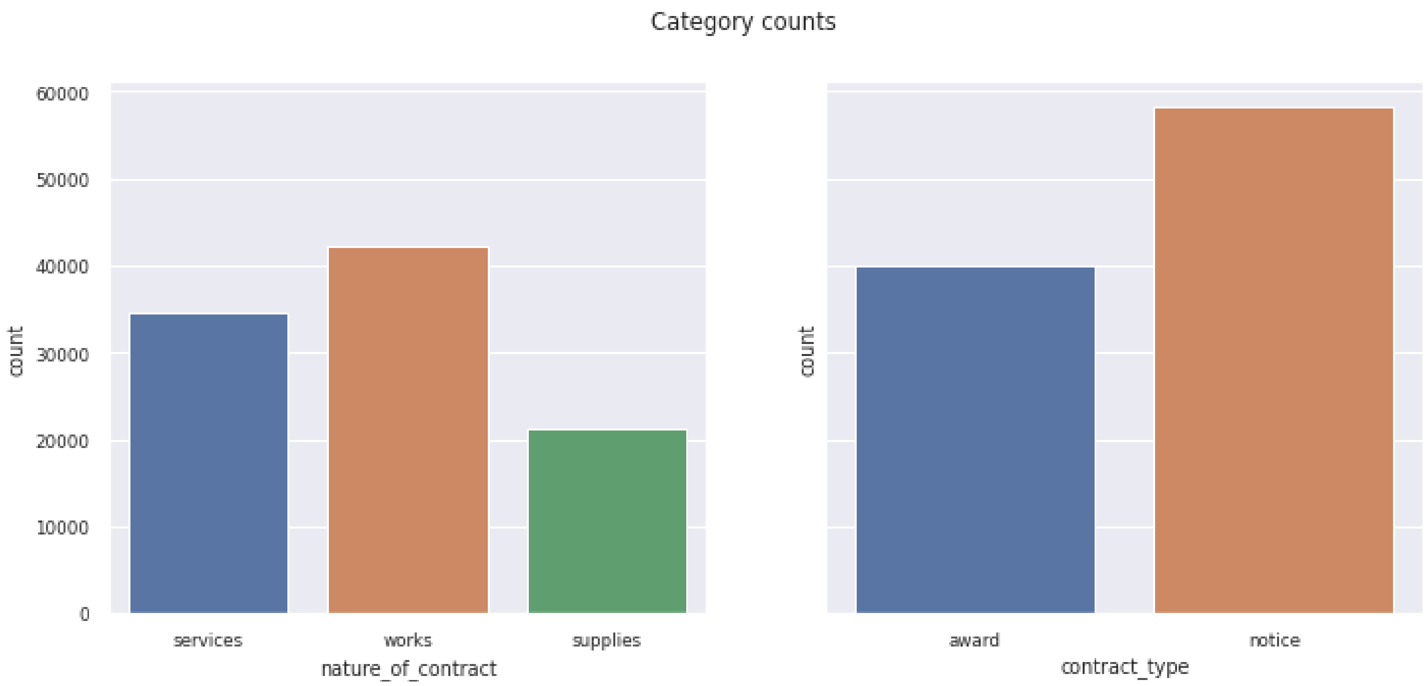
## References

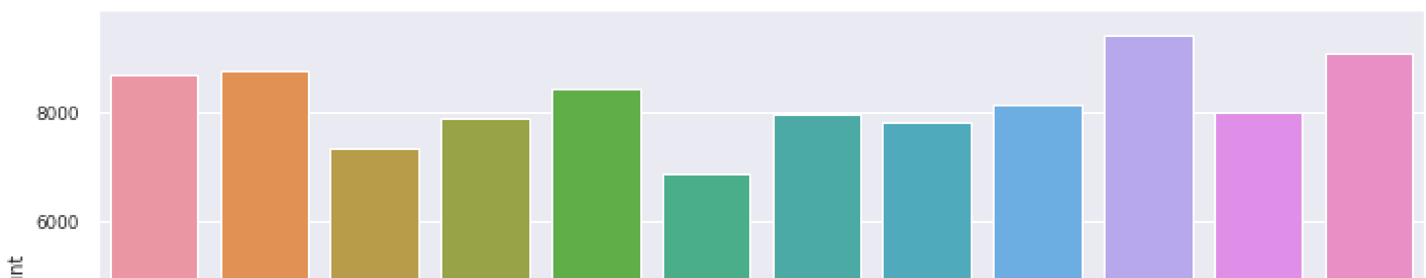
1. Li, S., 2019. Medium. [online] Medium. Available at: https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17 [Accessed 7 April 2021].
2. Howard, J., 2019. NB-SVM strong linear baseline. [online] Kaggle.com. Available at: https://www.kaggle.com/jhoward/nb-svm-strong-linear-baseline?fbclid=IwAR3wdjBD1AZHEJ4EJC0JOqsFJIqpnmrX9CNh2udISKihOliP31S0cXb6ggo [Accessed 5 April 2021].
3. Wang, S. and C.D. Manning (2012) "Baselines and Bigrams: Simple, Good Sentiment and Topic Classification". Department of Computer Science, Stanford University
4. Verma, S. (2019) [online] 'Multi-Label Image Classification with Neural Network | Keras'. https://towardsdatascience.com/multi-label-image-classification-with-neural-network-keras-ddc1ab1afede [Accessed 27 March 2021]
5. MathWorks (2021) [online] 'Multilabel Text Classification Using Deep Learning'. https://uk.mathworks.com/help/deeplearning/ug/multilabel-text-classification-using-deep-learning.html [Accessed 8 April 2021]

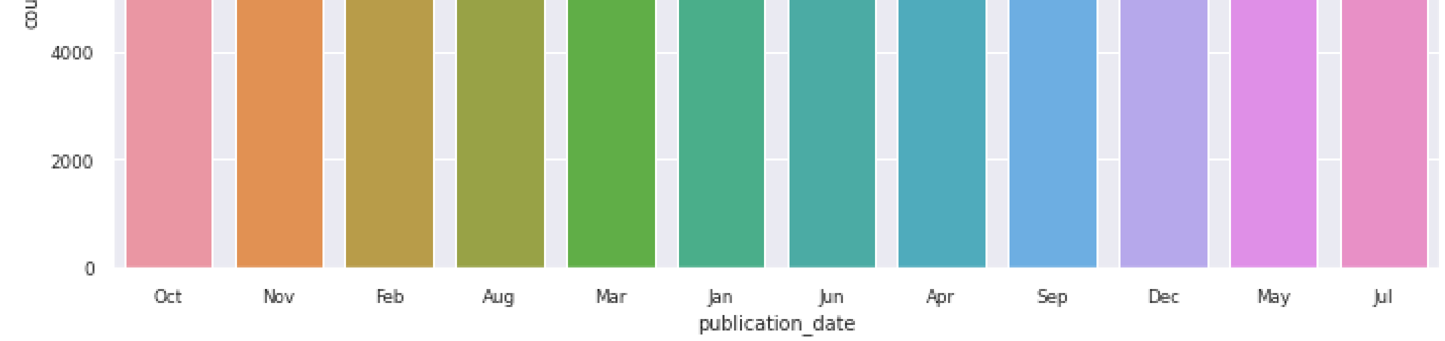
```
# This code will allow google drive to be mounted
from google.colab import drive
drive.mount('/content/drive')


# This code will allow EDA & pre-processing file to be ran within writeup file
# Please import the EDA-processing file as an attachment (in Colab)
# or use an appropriate file path
!pip install ipynb
%run /content/Task_2_EDA_Processing.ipynb
```

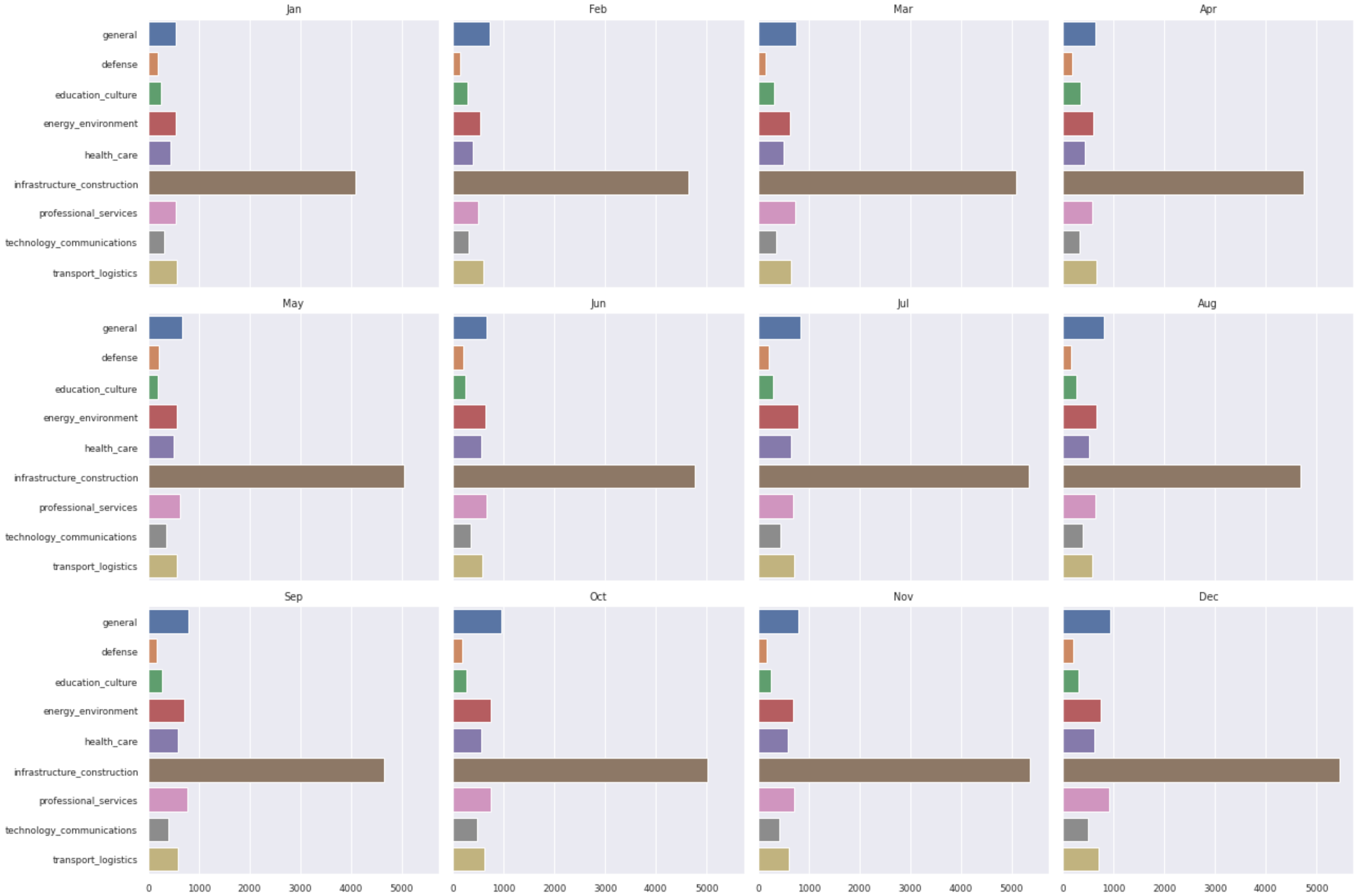
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/co
Requirement already satisfied: ipynb in /usr/local/lib/python3.7/dist-packages (0.5.1)
Requirement already satisfied: tensorflow_addons in /usr/local/lib/python3.7/dist-packages (0.
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-packages (from
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/co
docid : 98320
publication_date : 254
contract_type : 2
nature_of_contract : 3
country_code : 1
country_name : 1
sector : 1
category : 307
title : 32207
description : 70652
awarding_authority : 13191
label : 176
```






```
Average number of characters in each title is 51. The maximum and minimum lengths are 147 and
Average number of characters in each description is 306. The maximum and minimum lengths are 3
```

# Models

## Baseline Naive Bayes Classifier

```
def Bayes_Prob(y_i, y):
    prob = X_train[y==y_i].sum(0) #probability of y==y_i given x.
    return (prob+1) / ((y==y_i).sum()+1) #returns the vector dimension (1, 303654) of probabilit

def get_mdl(y):
    y = y.values #makes an X by 0 array of the values of the column being trained (i.e 'general')
    log_vector = np.log(Bayes_Prob(1,y) / Bayes_Prob(0,y)) #the log of the vectors where words ar
    model = LogisticRegression(C=4, dual=False)
    input_log_vector = X_train.multiply(log_vector) #combine the input vector with the log
    return model.fit(input_log_vector, y), log_vector #train the logistic regression of that colu
```

## Predictions on validation set

```
#Predictions - an array of 9 binary classifiers combining them to form a whole prediction per co
preds = np.zeros((y_valid.shape[0],len(label_cols)))
for i, j in enumerate(label_cols):
    print('fit', j)
    model,log_vector = get_mdl(y_train[j])
    preds[:,i] = model.predict_proba(X_valid.multiply(log_vector))[:,1] #adds the prediction to
```

```
fit general
fit defense
fit education_culture
fit energy_environment
fit health_care
fit infrastructure_construction
fit professional_services
fit technology_communications
fit transport_logistics
```

```
predictions = (np.array(preds) >= 0.5).astype(int)
labels = np.array(y_valid)

TP, TN, FP, FN = 0,0,0,0

for i in range(len(labels)):
    for j in range(labels.shape[1]):
        if predictions[i][j] == labels[i][j]: ##
            if predictions[i][j] == 1:
                TP +=1
            elif predictions [i][j] == 0:
                TN +=1
        elif predictions[i][j] != labels[i][j]:
            if predictions [i][j] == 1:
                FP +=1
            elif predictions [i][j] == 0:
                FN +=1


accuracy = (TP + TN) / (TP+TN+FP+FN)
recall = TP / (TP+FN)
precision = TP / (TP+FP)

print ("Accuracy: ",accuracy)
print ("Recall: ", recall)
print ("Precision: ",precision)
```

```
Accuracy:  0.9903588039491605
Recall:  0.9285217633152552
Precision:  0.99187917801436
```

**Metrics on validation set**:

Accuracy: 0.9906696279719015

Recall: 0.9315036686170707

Precision: 0.9913021991598715

## Prediction on test data

```
preds = np.zeros((test_doc.shape[0],len(label_cols)))
for i, j in enumerate(label_cols):
    print('fit', j)
    model,log_vector = get_mdl(y_train[j])
    preds[:,i] = model.predict_proba(test_doc.multiply(log_vector))[:,1]
```

```
fit general
fit defense
fit education_culture
fit energy_environment
fit health_care
fit infrastructure_construction
fit professional_services
fit technology_communications
fit transport_logistics
```

```
#Create a dataframe of solutions
submid = pd.DataFrame({'docid': test['docid']})
submission = pd.concat([submid, pd.DataFrame(np.round(preds), columns = label_cols)], axis=1) #Ro
```

```
#Combine solutions to one string, delete previous solutions in columns and append final string so
labels = []
for i in range(len(submission)):
    row = submission.iloc[i]
    info =  [int(row['general']),int(row['defense']), int(row['education_culture']), int(row['ene
    label= ''
    for i in info:
        label = label + str(i)
    labels.append(str(label))

submission['label'] = labels
submission.drop(['general','defense','education_culture','energy_environment','health_care', 'inf
submission.to_csv('nb_submission.csv', index=False)
```

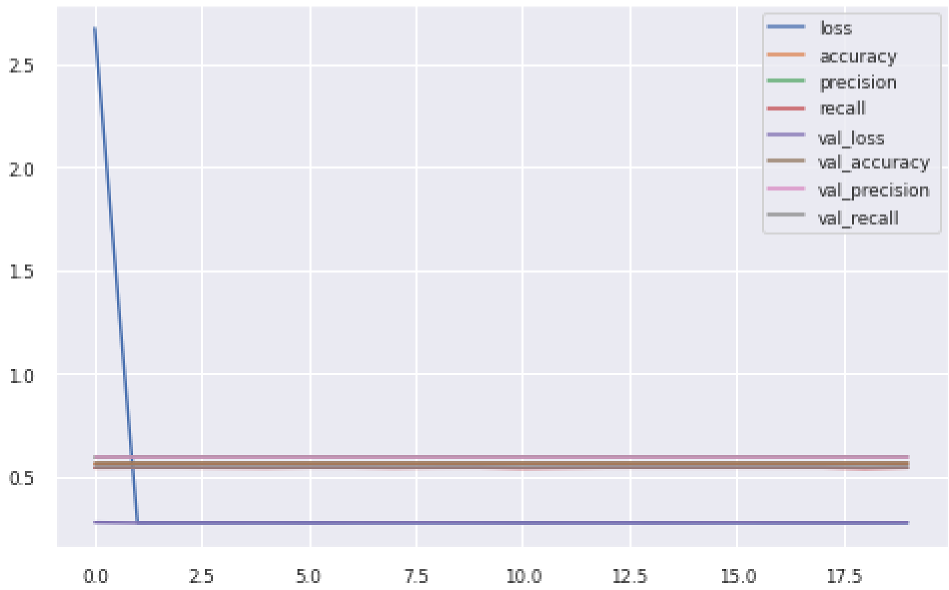
## Neural Network

```
def NN_builder(layers=1, nodes=1, deep=False, active='relu',output_func='sigmoid',dropout=False):
    input_doc = keras.layers.Input(shape=train_text.shape[1:])
    model = keras.models.Sequential()
    hidden=list()
    for i in range(layers):
        hidden.append('h%s' %i)
        if i==0:
            hidden[i]=keras.layers.Dense(nodes, activation=active, kernel_initializer='he_unifor
        else:
            hidden[i]=keras.layers.Dense(nodes, activation=active, kernel_initializer='he_unifor
    print(hidden[-1])
    if dropout==True:
        hiddendrop = keras.layers.Dropout(rate=0.2)(hidden[-1])
        if deep==True:
            concat = keras.layers.Concatenate()([input_doc,hiddendrop])
            output = keras.layers.Dense(9, activation=output_func)(hidden[-1])
        else:
            output = keras.layers.Dense(9, activation=output_func)(hidden[-1])
    else:
        if deep==True:
```

```
        if deep==True:
            concat = keras.layers.Concatenate()([input_doc,hidden[-1]])
            output = keras.layers.Dense(9, activation=output_func)(hidden[-1])
        else:
            output = keras.layers.Dense(9, activation=output_func)(hidden[-1])
    model = keras.Model(inputs=[input_doc], outputs=[output])
    return model
```

```
WnD_model=NN_builder(3,20,active='relu',output_func='sigmoid',deep=True, dropout=True)
WnD_model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.1),metrics=['accura
history = WnD_model.fit(train_text, train_labels, epochs=20, validation_data=(val_text, val_labe
err_test = WnD_model.evaluate(val_text, val_labels)

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
f1=2*(err_test[2]*err_test[3])/((err_test[2]+err_test[3]))
print("Loss {}\nAccuracy {}\nPrecision {}\nRecall {}\nF1 {}".format(err_test[0],err_test[1],err_t
```

```
KerasTensor(type_spec=TensorSpec(shape=(None, 20), dtype=tf.float32, name=None), name='dense_2
615/615 [==============================] - 2s 3ms/step - loss: 0.2791 - accuracy: 0.5657 - pre
Loss 0.27913036942481995
Accuracy 0.5657392740249634
Precision 0.5992065668106079
Recall 0.5460993051528931
F1 0.5714216573732047
```




Metrics

- **Loss**: 0.278877854347229
- **Accuracy**: 0.5653324127197266
- **Precision**: 0.6010884642601013
- **Recall**: 0.5448091626167297
- **F1**: 0.571566770348712
- **Kaggle**: Accuracy 0.10435

# Model A - LSTM

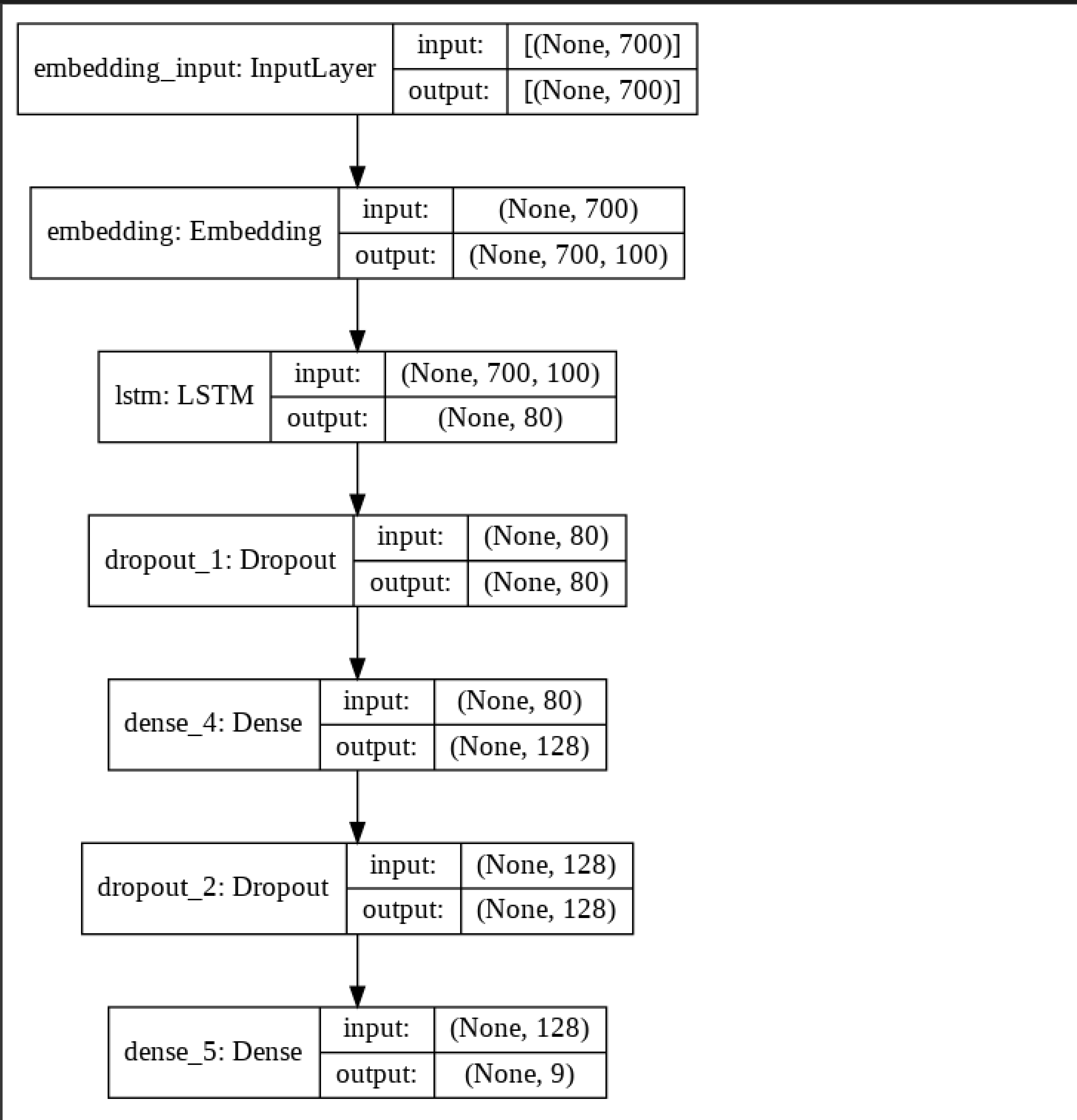
```
class NN_model():
    """
```

```
    This class was used to build and compile the two final neural networks.
    The default parameters provided turned out to give the best results.
    """
    def __init__(self, max_words=50000+1,
                 embedding_dim=100,
                 input_length=700,
                 kernel_initializer = 'he_normal',
                 hidden_activation = 'elu',
                 output_activation = 'sigmoid',
                 dropout=0.5,
                 loss = 'binary_crossentropy',
                 optimizer = 'adam'):

        self.max_words = max_words
        self.embedding_dim = embedding_dim
        self.input_length = input_length
        self.kernel_initializer = kernel_initializer
        self.hidden_activation = hidden_activation
        self.output_activation = output_activation
        self.dropout = dropout
        self.loss = loss
        self.optimizer = optimizer


    def build_model_A(self):
      model = Sequential()
      model.add(Embedding(self.max_words, self.embedding_dim, input_length=self.input_length))
      model.add(LSTM(80))
      model.add(Dropout(self.dropout))
      model.add(Dense(128, kernel_initializer=self.kernel_initializer, activation=self.hidden_act
      model.add(Dropout(self.dropout))
      model.add(Dense(9, activation=self.output_activation))
      model.compile(loss= self.loss, optimizer=self.optimizer, metrics=['acc', tf.keras.metrics.P
      return model


    def build_model_B(self):
      input_1 = Input(shape=(self.input_length))
      input_2 = Input(shape=(14,))

      embedding_layer = Embedding(self.max_words, self.embedding_dim, input_length=self.input_len
      GRU_Layer_1 = GRU(128)(embedding_layer)
      dropout_layer_1 = Dropout(self.dropout)(GRU_Layer_1)

      dense_layer_1 = Dense(128, kernel_initializer= self.kernel_initializer, activation= self.hi
      dense_layer_2 = Dense(128, kernel_initializer= self.kernel_initializer, activation= self.hi
      dropout_layer_2 = Dropout(self.dropout)(dense_layer_2)

      concat_layer = Concatenate()([dropout_layer_1, dropout_layer_2])
      dense_layer_3 = Dense(64, kernel_initializer= self.kernel_initializer, activation= self.hid
      dropout_layer_3 = Dropout(self.dropout)(dense_layer_3)

      output = Dense(9, activation= self.output_activation)(dropout_layer_3)
      multi_input = Model(inputs=[input_1, input_2], outputs=output)
      multi_input.compile(loss= self.loss, optimizer=self.optimizer, metrics=['acc', tf.keras.met
      return multi_input
```

```
EPOCHS = 20
```

```
BATCH_SIZE = 40
```

```
net = NN_model()
model_a = net.build_model_A()
plot_model(model_a, to_file='model_plot3.png', show_shapes=True, show_layer_names=True)
```



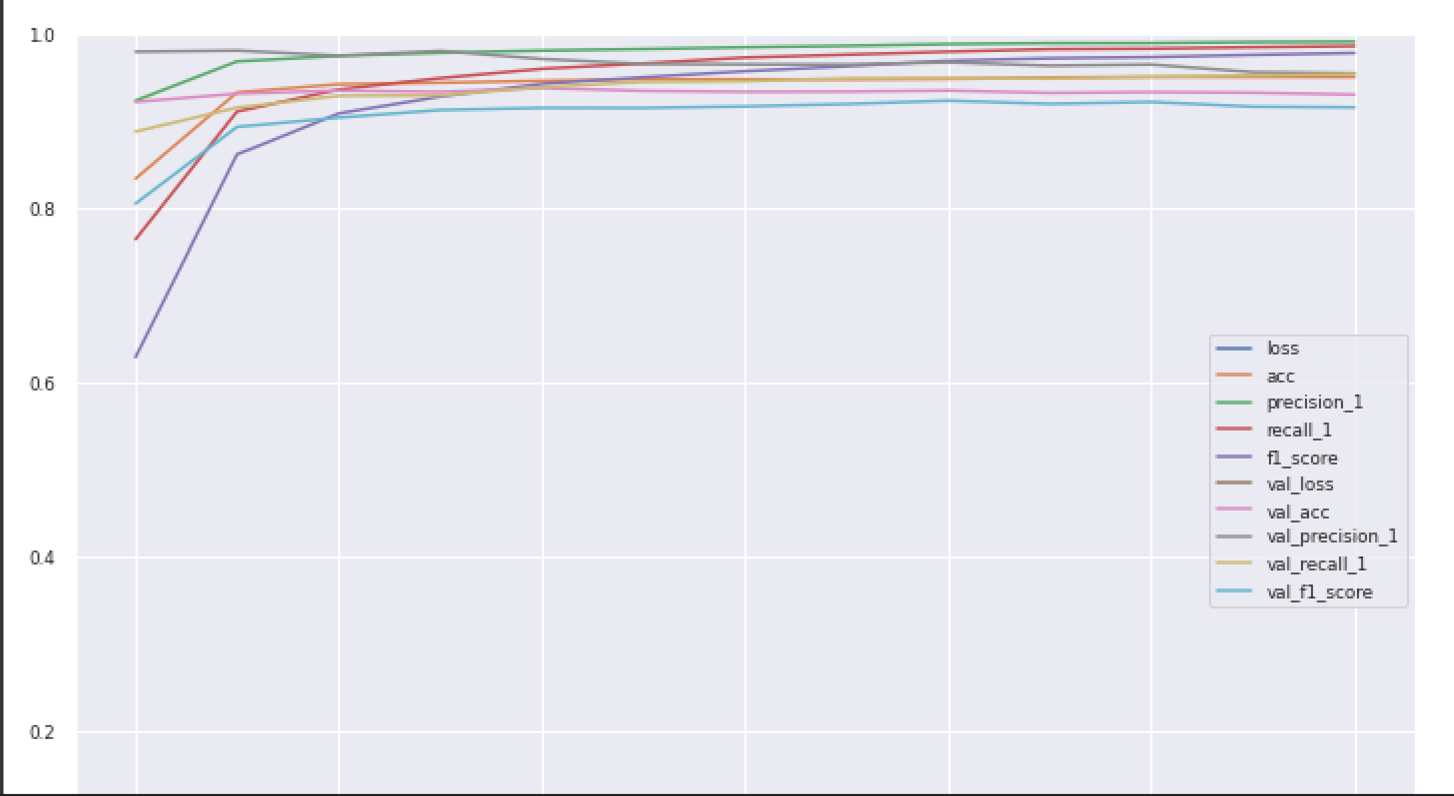

```
# stops early if no improvement
early_stop_a = EarlyStopping(monitor='val_f1_score', mode='max', verbose=1, patience=4)
best_model_a = ModelCheckpoint('Model_a_optimal_accuracy.h5', monitor='val_f1_score', mode='max'
```

```
# Replace val split with validation x and y sets
history = model_a.fit(train_text, train_labels,
                    epochs=EPOCHS, batch_size=BATCH_SIZE,
                    callbacks=[early_stop_a, best_model_a],
                    validation_data=(val_text, val_labels))
```

```
Epoch 1/20
```

```
1967/1967 [==============================] - 147s 66ms/step - loss: 0.1859 - acc: 0.7223 - pre

Epoch 00001: val_f1_score improved from -inf to 0.80590, saving model to Model_a_optimal_accur
Epoch 2/20
1967/1967 [==============================] - 130s 66ms/step - loss: 0.0531 - acc: 0.9319 - pre

Epoch 00002: val_f1_score improved from 0.80590 to 0.89418, saving model to Model_a_optimal_ac
Epoch 3/20
1967/1967 [==============================] - 131s 67ms/step - loss: 0.0374 - acc: 0.9450 - pre

Epoch 00003: val_f1_score improved from 0.89418 to 0.90485, saving model to Model_a_optimal_ac
Epoch 4/20
1967/1967 [==============================] - 130s 66ms/step - loss: 0.0287 - acc: 0.9465 - pre

Epoch 00004: val_f1_score improved from 0.90485 to 0.91354, saving model to Model_a_optimal_ac
Epoch 5/20
1967/1967 [==============================] - 133s 67ms/step - loss: 0.0230 - acc: 0.9453 - pre

Epoch 00005: val_f1_score improved from 0.91354 to 0.91602, saving model to Model_a_optimal_ac
Epoch 6/20
1967/1967 [==============================] - 132s 67ms/step - loss: 0.0188 - acc: 0.9476 - pre

Epoch 00006: val_f1_score did not improve from 0.91602
Epoch 7/20
1967/1967 [==============================] - 133s 68ms/step - loss: 0.0155 - acc: 0.9486 - pre

Epoch 00007: val_f1_score improved from 0.91602 to 0.91772, saving model to Model_a_optimal_ac
Epoch 8/20
1967/1967 [==============================] - 136s 69ms/step - loss: 0.0141 - acc: 0.9490 - pre

Epoch 00008: val_f1_score improved from 0.91772 to 0.92044, saving model to Model_a_optimal_ac
Epoch 9/20
1967/1967 [==============================] - 134s 68ms/step - loss: 0.0114 - acc: 0.9509 - pre

Epoch 00009: val_f1_score improved from 0.92044 to 0.92433, saving model to Model_a_optimal_ac
Epoch 10/20
1967/1967 [==============================] - 136s 69ms/step - loss: 0.0097 - acc: 0.9506 - pre

Epoch 00010: val_f1_score did not improve from 0.92433
Epoch 11/20
1967/1967 [==============================] - 136s 69ms/step - loss: 0.0098 - acc: 0.9518 - pre

Epoch 00011: val_f1_score did not improve from 0.92433
Epoch 12/20
1967/1967 [==============================] - 134s 68ms/step - loss: 0.0086 - acc: 0.9518 - pre

Epoch 00012: val_f1_score did not improve from 0.92433
Epoch 13/20
1967/1967 [==============================] - 132s 67ms/step - loss: 0.0079 - acc: 0.9523 - pre

Epoch 00013: val_f1_score did not improve from 0.92433
Epoch 00013: early stopping
```

```python
pd.DataFrame(history.history).plot(figsize=(12, 8))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```




```
best_model_a = load_model("Model_a_optimal_accuracy.h5")
```

```
def convert_to_string(predictions):
    """
    Converts the 9-digit array of floats back into
    a binary string.
    """
    # Rounds to 0 or 1
    result_rounded = (predictions.copy() >= 0.5).astype(int)
    final_predictions = []

    for prediction in result_rounded:
        some_string = ''
        for num in prediction:
            some_string += str(num)
        final_predictions.append(some_string)

    return final_predictions
```

```
result = best_model_a.predict(test_text)
final_predictions = convert_to_string(result)
```

```
submid = pd.DataFrame({'docid': test['docid'], 'label':final_predictions}).to_csv('upload.csv_a'
```

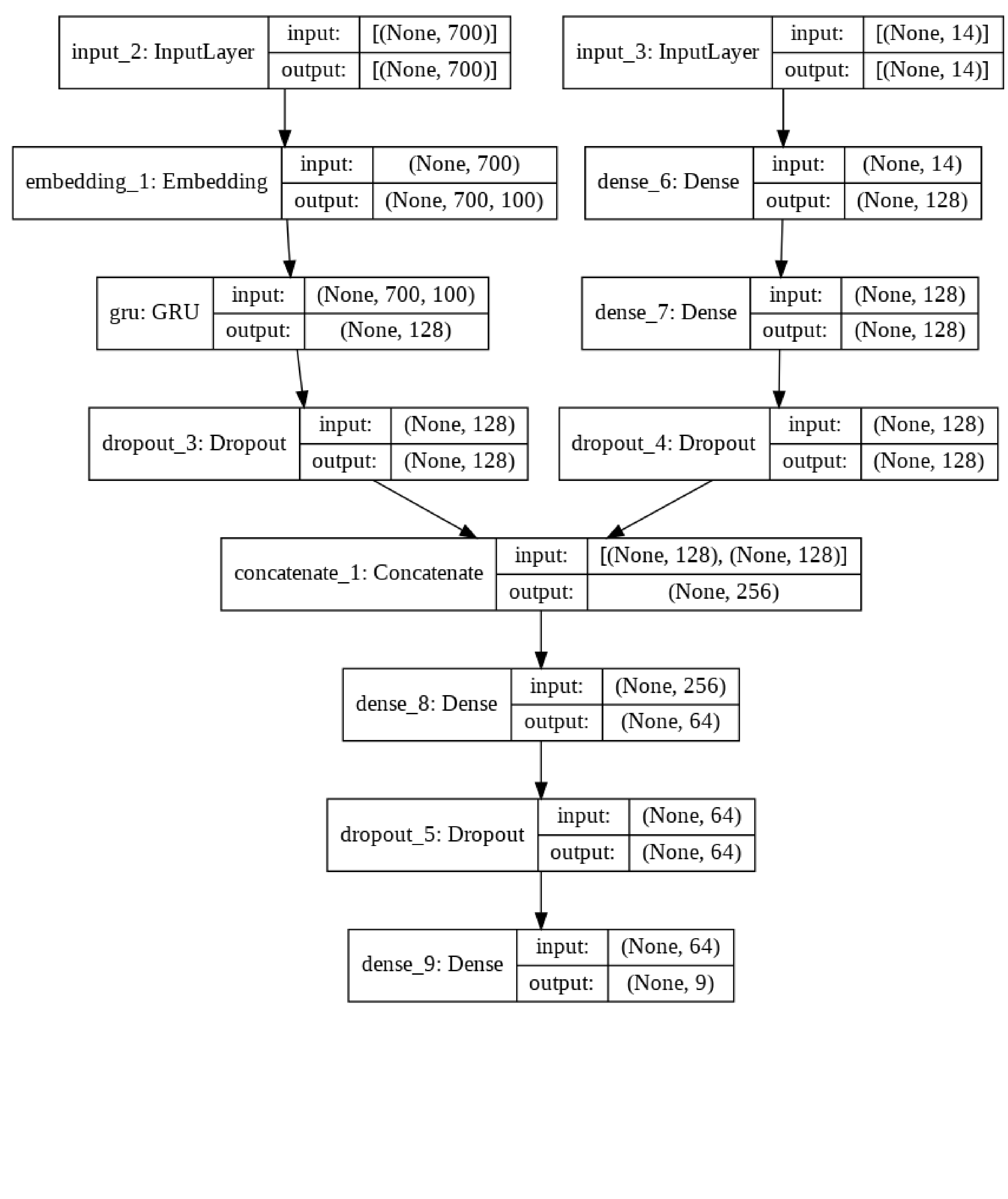
Model A test performance = 0.93261

## Multi-input Deep Learner B

The below model takes both text and categorical inputs to be run through different layers better suited to each data type before contatentating their ouputs for a final prediction.

```
model_b = net.build_model_B()
```

```
plot_model(model_b, show_shapes=True, show_layer_names=True)
```




```
# stops early if no improvement
early_stop_b = EarlyStopping(monitor='val_f1_score', mode='max', verbose=1, patience=4)
best_model_b = ModelCheckpoint('Model_b_optimal_accuracy.h5', monitor='val_f1_score', mode='max',
```

```
history = model_b.fit(x=[train_text, train_categoricals], y=train_labels,
                      batch_size=BATCH_SIZE, epochs=EPOCHS, verbose=1, callbacks=[early_stop_
                      validation_data=((val_text, val_categoricals), val_labels))
```

```
Epoch 1/20
1967/1967 [==============================] - 130s 65ms/step - loss: 0.2001 - acc: 0.6980 - pre
```

```
Epoch 00001: val_f1_score improved from -inf to 0.79755, saving model to Model_b_optimal_accur
Epoch 2/20
1967/1967 [==============================] - 130s 66ms/step - loss: 0.0642 - acc: 0.9291 - pre

Epoch 00002: val_f1_score improved from 0.79755 to 0.89020, saving model to Model_b_optimal_ac
Epoch 3/20
1967/1967 [==============================] - 130s 66ms/step - loss: 0.0462 - acc: 0.9427 - pre

Epoch 00003: val_f1_score improved from 0.89020 to 0.90285, saving model to Model_b_optimal_ac
Epoch 4/20
1967/1967 [==============================] - 128s 65ms/step - loss: 0.0350 - acc: 0.9456 - pre

Epoch 00004: val_f1_score improved from 0.90285 to 0.91076, saving model to Model_b_optimal_ac
Epoch 5/20
1967/1967 [==============================] - 127s 65ms/step - loss: 0.0291 - acc: 0.9462 - pre

Epoch 00005: val_f1_score improved from 0.91076 to 0.91425, saving model to Model_b_optimal_ac
Epoch 6/20
1967/1967 [==============================] - 127s 65ms/step - loss: 0.0243 - acc: 0.9472 - pre

Epoch 00006: val_f1_score improved from 0.91425 to 0.91676, saving model to Model_b_optimal_ac
Epoch 7/20
1967/1967 [==============================] - 126s 64ms/step - loss: 0.0204 - acc: 0.9460 - pre

Epoch 00007: val_f1_score improved from 0.91676 to 0.92067, saving model to Model_b_optimal_ac
Epoch 8/20
1967/1967 [==============================] - 127s 65ms/step - loss: 0.0175 - acc: 0.9493 - pre

Epoch 00008: val_f1_score did not improve from 0.92067
Epoch 9/20
1967/1967 [==============================] - 125s 63ms/step - loss: 0.0154 - acc: 0.9477 - pre

Epoch 00009: val_f1_score improved from 0.92067 to 0.92281, saving model to Model_b_optimal_ac
Epoch 10/20
1967/1967 [==============================] - 125s 63ms/step - loss: 0.0144 - acc: 0.9483 - pre

Epoch 00010: val_f1_score improved from 0.92281 to 0.92485, saving model to Model_b_optimal_ac
Epoch 11/20
1967/1967 [==============================] - 125s 63ms/step - loss: 0.0133 - acc: 0.9471 - pre

Epoch 00011: val_f1_score did not improve from 0.92485
Epoch 12/20
1967/1967 [==============================] - 125s 63ms/step - loss: 0.0124 - acc: 0.9473 - pre

Epoch 00012: val_f1_score did not improve from 0.92485
Epoch 13/20
1967/1967 [==============================] - 124s 63ms/step - loss: 0.0113 - acc: 0.9482 - pre

Epoch 00013: val_f1_score did not improve from 0.92485
Epoch 14/20
1967/1967 [==============================] - 124s 63ms/step - loss: 0.0108 - acc: 0.9486 - pre

Epoch 00014: val_f1_score did not improve from 0.92485
Epoch 00014: early stopping
```

```
pd.DataFrame(history.history).plot(figsize=(12, 8))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```

```
best_model_b = load_model("Model_b_optimal_accuracy.h5")
```

```python
result = best_model_b.predict((test_text, test_categoricals))
final_predictions = convert_to_string(result)
```

```python
pd.DataFrame({'docid': test['docid'], 'label':final_predictions}).to_csv('upload_b.csv', index=Fa
```

Model B Kaggle performance = 0.93790

8m 31s completed at 12:45