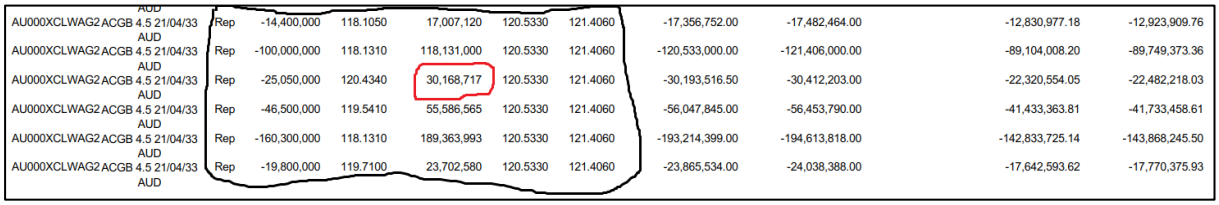
An alternative approach for the value-based pipeline (especially for tabular data)

By Biswapratap Chatterjee

The problem

The existing value-based pipeline uses contextual and location as embeddings of boxes/regions of a PDF document for classification using a deep and wide neural network model. Although, this approach has shown substantially high accuracy in few specific genres of documents like the “K1 Forms” (where it has a relatively consistent and simple pattern of text organization), it has failed for complex and inconsistent documents, like Broker Statements. Manual analysis has showed the following problems –

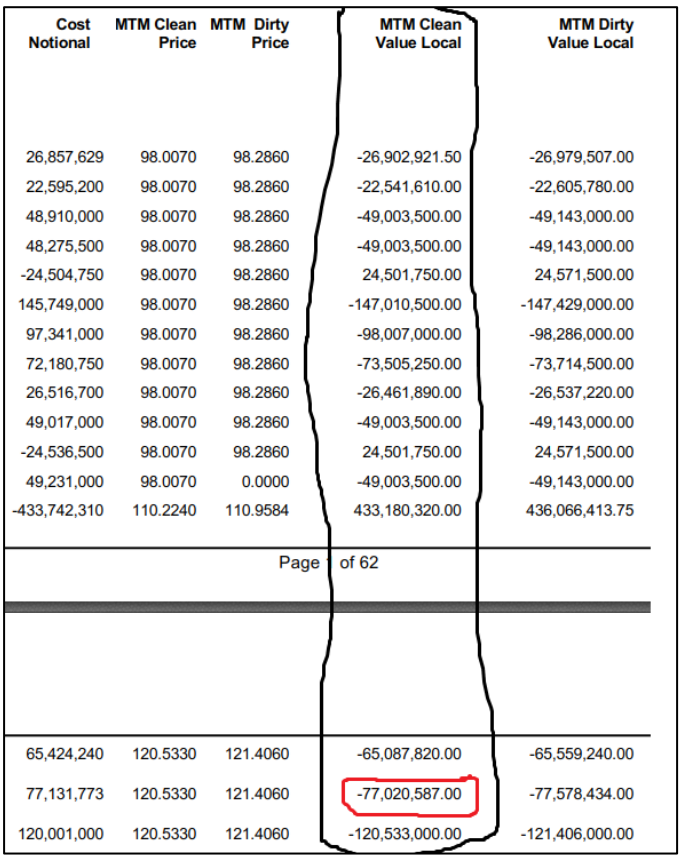
1. Since there is a lot of tabular data which spans across many pages, so a left, top, right, bottom context window loses its significance most of the time, as shown below –



AU000XCLWAG2 ACGB 4.5 21/04/33 AUD	Rep	-14,400,000	118.1050	17,007,120	120.5330	121.4060	-17,356,752.00	-17,482,464.00	-12,830,977.18	-12,923,909.76
AU000XCLWAG2 ACGB 4.5 21/04/33 AUD	Rep	-100,000,000	118.1310	118,131,000	120.5330	121.4060	-120,533,000.00	-121,406,000.00	-89,104,008.20	-89,749,373.36
AU000XCLWAG2 ACGB 4.5 21/04/33 AUD	Rep	-25,050,000	120.4340	30,168,717	120.5330	121.4060	-30,193,516.50	-30,412,203.00	-22,320,554.05	-22,482,218.03
AU000XCLWAG2 ACGB 4.5 21/04/33 AUD	Rep	-46,500,000	119.5410	55,586,565	120.5330	121.4060	-56,047,845.00	-56,453,790.00	-41,433,363.81	-41,733,458.61
AU000XCLWAG2 ACGB 4.5 21/04/33 AUD	Rep	-160,300,000	118.1310	189,363,993	120.5330	121.4060	-193,214,399.00	-194,613,818.00	-142,833,725.14	-143,868,245.50
AU000XCLWAG2 ACGB 4.5 21/04/33 AUD	Rep	-19,800,000	119.7100	23,702,580	120.5330	121.4060	-23,865,534.00	-24,038,388.00	-17,642,593.62	-17,770,375.93

As you can see the context windows (in black) for the item 30,168,717 (in red) does not hold any significance. There are all numbers around it and the embeddings cannot abstract what those numbers are.

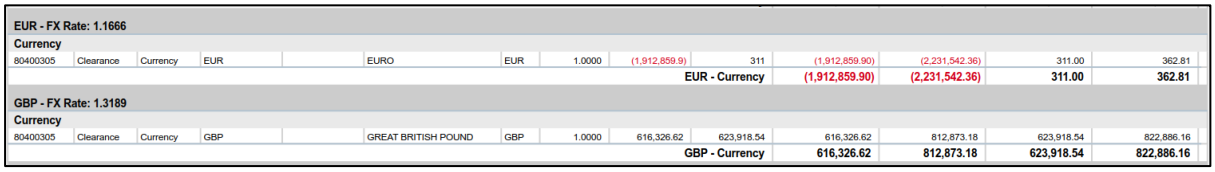
2. Again, due to the tabular nature of the data, the actual meaningful context of any box may remain very far away from the box (only when all pages are concatenated as a single page). And if we split the data into pages than the meaningful context is entirely lost, as shown below –



Cost Notional	MTM Clean Price	MTM Dirty Price	MTM Clean Value Local	MTM Dirty Value Local
26,857,629	98.0070	98.2860	-26,902,921.50	-26,979,507.00
22,595,200	98.0070	98.2860	-22,541,610.00	-22,605,780.00
48,910,000	98.0070	98.2860	-49,003,500.00	-49,143,000.00
48,275,500	98.0070	98.2860	-49,003,500.00	-49,143,000.00
-24,504,750	98.0070	98.2860	24,501,750.00	24,571,500.00
145,749,000	98.0070	98.2860	-147,010,500.00	-147,429,000.00
97,341,000	98.0070	98.2860	-98,007,000.00	-98,286,000.00
72,180,750	98.0070	98.2860	-73,505,250.00	-73,714,500.00
26,516,700	98.0070	98.2860	-26,461,890.00	-26,537,220.00
49,017,000	98.0070	98.2860	-49,003,500.00	-49,143,000.00
-24,536,500	98.0070	98.2860	24,501,750.00	24,571,500.00
49,231,000	98.0070	0.0000	-49,003,500.00	-49,143,000.00
-433,742,310	110.2240	110.9584	433,180,320.00	436,066,413.75
		Page 1 of 62		
65,424,240	120.5330	121.4060	-65,087,820.00	-65,559,240.00
77,131,773	120.5330	121.4060	-77,020,587.00	-77,578,434.00
120,001,000	120.5330	121.4060	-120,533,000.00	-121,406,000.00

As you can see here the actual meaningful context of the item -77,020,587.00 (in red) is located very far away from it i.e. the column name “MTM Clean Value Local”, and in this case even separated by an entire page.

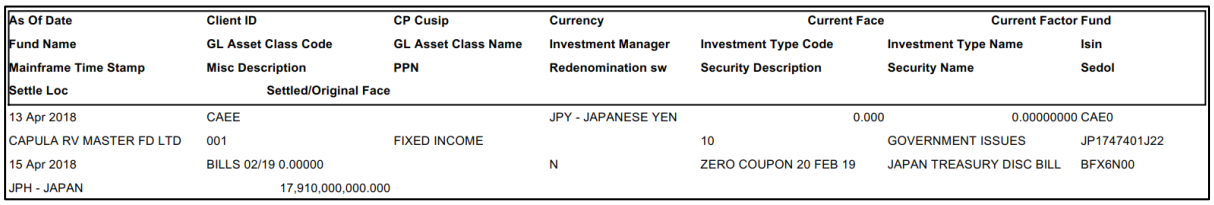
3. Drastic changes in data organization structure for similar data across clients, like –



EUR - FX Rate: 1.1666													
Currency													
80400305	Clearance	Currency	EUR		EURO	EUR	1.0000	(1,912,859.9)	311	(1,912,859.90)	(2,231,542.36)	311.00	362.81
									EUR - Currency	**(1,912,859.90)**	**(2,231,542.36)**	**311.00**	**362.81**
GBP - FX Rate: 1.3189													
Currency													
80400305	Clearance	Currency	GBP		GREAT BRITISH POUND	GBP	1.0000	616,326.62	623,918.54	616,326.62	812,873.18	623,918.54	822,886.16
									GBP - Currency	**616,326.62**	**812,873.18**	**623,918.54**	**822,886.16**

As you can see here the tabular data organization structure is so different in this case. If there are few exceptions than they can be treated as outliers by the agent, but in case of broker statements there are many such variations.

4. Another major problem is due to lack of right margin space many times the tabular data wraps around to the next line, which becomes very difficult to identify by the agent, like –



As Of Date	**Client ID**	**CP Cusip**	**Currency**	**Current Face**	**Current Factor**	**Fund**
Fund Name	**GL Asset Class Code**	**GL Asset Class Name**	**Investment Manager**	**Investment Type Code**	**Investment Type Name**	**Isin**
Mainframe Time Stamp	**Misc Description**	**PPN**	**Redenomination sw**	**Security Description**	**Security Name**	**Sedol**
Settle Loc	**Settled/Original Face**					
13 Apr 2018	CAEE		JPY - JAPANESE YEN	0.000	0.00000000	CAE0
CAPULA RV MASTER FD LTD	001	FIXED INCOME		10	GOVERNMENT ISSUES	JP1747401J22
15 Apr 2018	BILLS 02/19 0.00000		N	ZERO COUPON 20 FEB 19	JAPAN TREASURY DISC BILL	BFX6N00
JPH - JAPAN	17,910,000,000.000					

5. An indirect problem is the nature of the embeddings themselves. We believe the broker statements, invoices or any such information source which is tabular and discreet in nature doesn’t fit well with the concept of context in general. Textual context makes more sense in a corpus which is more subjective in nature like a – news report, a book, an article etc. Consider the most common example of word2vec embeddings:

   $Vector_{Queen} = Vector_{King} - Vector_{Man}$

   As we can see here there is a definitive relationship between the vectors of the word embeddings, and it is possible only because the corpus from which these embeddings are generated are subjective in nature which talks a lot about – Who is a King? Who is a Queen? And Who is a Man?

   But if we see the information inside a broker statement (consider the one in point 4), we as humans even can not predict the relationship between $Vector_{Current\ Face}$ and $Vector_{Security\ Description}$, because there is just not enough literature explaining the meaning of the phrases.

Proposed Solution

The proposed solution is an entirely novel approach. Instead of using contextual embeddings we go for sequential embeddings. The sequential embeddings are different from the location embeddings used in the existing approach as the order of the sequence is important. Why?

Consider the example in Problem point 2. As a human we understand the meaning of the text -77,020,587.00 as an MTM Clean Value Local because when trace upwards in a straight line we find this to be the column name. Now once we know this and the same with its neighbouring texts, we can understand the meaning of the text -120,533,000.00 without even tracing upwards, we can understand it by the left sequence order of -<Number>→<Number>→<Number>??

Hypothesis 1

So, as you can see the context of each word in the document has a stronger significance about its meaning w.r.t its sequential order rather than a context window. **This hypothesis demands for a LSTM ANN, because we are so much focused on the order of the sequences rather than a window of sequences.**

Hypothesis 2

There is an implicit problem here. How far a sequence should be to guarantee a significant extraction of meaning? The answer is we can never have a definitive answer for this question. So, we always take the maximum sequence possible. But this introduces another challenge as LSTMs are not very good at remembering very long term embeddings. **This is where we introduce an Attention ANN.** A LSTM coupled with an Attention NN significantly improves the ability of the LSTM to remember very long term embeddings in the sequence which has stronger effect on the target embedding.

Algorithm

Step 1: Read all the Abbey processed broker statement files and build a corpus.

```
def build_corpus(self, df):
    df = df[df['Map_Name'] != 'irrelevant']
    df = df[np.invert(df['Map_Name'].str.contains("/([_noclass|_noclass_|noclass_])/g",
regex=True))]
    if self.use_trim:
        text_list = df['Map_Result_Text'].tolist()
        doc_text = ''
        for text in text_list:
            try:
                text = text[2:-1].lower()
                if self.has_alphabet(text) or self.has_number(text):
                    doc_text = doc_text + text + ' '
            except Exception as e:
                pass
        self.docs.append(df)
        self.corpus.append(doc_text)
```

Step 2: While building the corpus, remove the rows which has labelled classes as – irrelevant or composite classes and if their associated texts do not contain any valid number or alphabets.

Step 3: Initialize a TFIDF vectorizer with a user configured min threshold i.e. automatically trim all words below the min threshold percent value. So, if min threshold is 0.10 then remove the last 10% of unimportant words. The following APIs are used for the same –

```
self.vectorizer = TfidfVectorizer(stop_words='english', min_df=self.trim_th)
def get_valid_text(self):
    X = self.vectorizer.fit_transform(self.corpus)
    self.valid_words = self.vectorizer.get_feature_names()
def clean_up(self):
    annotations_df = pd.DataFrame()
    for df in self.docs:
        map_result_text_list = df['Map_Result_Text'].tolist()
        boolean_list = [True for _ in range(0, df.shape[0], 1)]
        for idx, t in enumerate(map_result_text_list):
            t = t[2:-1].lower()
            t_list = t.split()
            if set(t_list).issubset(set(self.valid_words)):
                boolean_list[idx] = True
            else:
                boolean_list[idx] = False
        try:
            df = df[boolean_list]
        except Exception as e:
            print(str(e))
        df = self.merge_pages(df)
        annotations_df = pd.concat([annotations_df, df], axis=0)
    try:
        label_encoder = LabelEncoder().fit(annotations_df['Map_Name'])
        labels = label_encoder.transform(annotations_df['Map_Name'])
        print(max(labels))
        annotations_df['Map_Name_Text'] = annotations_df['Map_Name']
        annotations_df['Map_Name'] = labels
    except Exception as e:
        label_encoder = None
        print(str(e))
    return label_encoder, annotations_df
```

Step 4: As you can figure out from the code snippets above, along with trimming the bottom 10% of unimportant words the cleanup API is also label encoding the labelled classes for the model, which will later be converted into one hot encoded vectors.

Step 5: Now the boxes identified by Abbey are tagged and labelled into a list of unique integer values.

```
texts = pd.Series([s[2:-1] for s in annotations_df['Map_Result_Text'].tolist()])
sg = SequenceGeneration(texts)
texts = sg.tag_series()
box_int_encoded_text = self.encode(texts)
```

Step 6: Now we must generate the ordered sequences of each labelled box. Remember, this is not a window of contexts but a window of ordered sequences.

Step 7: Although the data is tabular in nature, but still due to the risks of wrapping columns names and left/right/middle orientation of texts we need to define a realistic boundary for each box, for which we will need the following, how are they used will be explained later –

```
avg_height = np.mean(annotations_df['Value_Canvas_Region_Height'])
avg_width = np.mean(annotations_df['Value_Canvas_Region_Width'])
```

Step 8: Now we need to calculate the ordered sequences for each box from the LEFT, TOP, RIGHT and BOTTOM. Below are the code snippets for each –

First filter the master DF for the json_id under question.

```
annotations_df_filtered = annotations_df[annotations_df['fileId'] == json_id]
```

Step 9: Then for LEFT sequence, notice how x_start, y_start and y_end is adjusted with the average width and height to expand into a realistic range. Finally, the filtered DF is paired with its X coordinated and named as left_tuple_seq.

```
x_start = box_x - (avg_width / 2)
y_start = box_y - (avg_height / 2)
y_end = box_y + height + (avg_height / 2)
annotations_df_filtered = annotations_df[annotations_df['Value_Canvas_Region_X'] <= x_start]
annotations_df_filtered = annotations_df_filtered[
    (annotations_df_filtered['Value_Canvas_Region_Y'] >= y_start) &
    (annotations_df_filtered['Value_Canvas_Region_Y'] <= y_end)]
left_tuple_seq = annotations_df_filtered[
    [ENCODED_COL_NAME, 'Value_Canvas_Region_X']].apply(tuple, axis=1).tolist()
```

Step 10: Similarly, repeat Step 9 for TOP, RIGHT and BOTTOM sequences, as shown below –

```
y_start = box_y - (avg_height / 2)
x_start = box_x - (avg_width / 2)
x_end = box_x + width + (avg_width / 2)
annotations_df_filtered = annotations_df[annotations_df['Value_Canvas_Region_Y'] <= y_start]
annotations_df_filtered = annotations_df_filtered[
    (annotations_df_filtered['Value_Canvas_Region_X'] >= x_start) &
    (annotations_df_filtered['Value_Canvas_Region_X'] <= x_end)]
top_tuple_seq = annotations_df_filtered[
    [ENCODED_COL_NAME, 'Value_Canvas_Region_Y']].apply(tuple, axis=1).tolist()
```

```
start_x = box_x + width - (avg_width / 2)
y_start = box_y - (avg_height / 2)
y_end = box_y + height + (avg_height / 2)
annotations_df_filtered = annotations_df[annotations_df['Value_Canvas_Region_X'] >= start_x]
annotations_df_filtered = annotations_df_filtered[
    (annotations_df_filtered['Value_Canvas_Region_Y'] >= y_start) &
    (annotations_df_filtered['Value_Canvas_Region_Y'] <= y_end)]
right_tuple_seq = annotations_df_filtered[
    [ENCODED_COL_NAME, 'Value_Canvas_Region_X']].apply(tuple, axis=1).tolist()
```

```
y_start = box_y + height - (avg_height / 2)
x_start = box_x - (avg_width / 2)
x_end = box_x + width + (avg_width / 2)
annotations_df_filtered = annotations_df[annotations_df['Value_Canvas_Region_Y'] >= y_start]
annotations_df_filtered = annotations_df_filtered[
    (annotations_df_filtered['Value_Canvas_Region_X'] >= x_start) &
    (annotations_df_filtered['Value_Canvas_Region_X'] <= x_end)]
bottom_tuple_seq = annotations_df_filtered[
    [ENCODED_COL_NAME, 'Value_Canvas_Region_Y']].apply(tuple, axis=1).tolist()
```

Step 11: Now we need to sort all the left/top/right/bottom_tuple_sequences w.r.t their RHS component i.e. either the X coordinates or the Y coordinates respectively. And then concatenate back to back all the ordered sequences in the following order – Left →Top →Right →Bottom.

```
def regularize_seq(self, tuple_seq):
    tuple_seq = sorted(tuple_seq, key=lambda x: x[1])
    seq = list()
    if len(tuple_seq) > 0:
        for lis in tuple_seq:
            embedding = lis[0]
            if isinstance(embedding, str):
                embedding = int(embedding)
            seq.append(embedding)
        return seq
    else:
        embedding = 0
        seq.append(0)
        return seq
```

```
seq_embedding = list()
seq_embedding.extend(l_seq)
seq_embedding.extend(t_seq)
seq_embedding.extend(r_seq)
seq_embedding.extend(b_seq)
```

Step 12: Finally, we need to find the max length of the sequence embeddings and pad zeros to the remaining sequence embeddings to bring them all to the same size.

```
max_seq_len = int(max(sequence_lengths))
print("Max Sequence Length: " + str(max_seq_len))
for idx, seq_embd in enumerate(seq_embedding_list):
    if len(seq_embd) < max_seq_len:
        pad_len = int(max_seq_len - len(seq_embd))
        for _ in range(0, pad_len, 1):
            seq_embd.append(0)
    elif len(seq_embd) > max_seq_len:
        trim_len = int(len(seq_embd) - max_seq_len)
        seq_embd = seq_embd[0:-trim_len]
    seq_embedding_list[idx] = seq_embd
normed_seq_embedding_matrix = np.array(seq_embedding_list)
```

Step 13: Now, we need to design our Model. As you can see, it's a bidirectional LSTM (because sequence orders on the past and future both are important) coupled with an Attention layer to help the bidirectional LSTM remember very old embeddings which has higher effect on the embedding vector in question. Additionally, as you can see, we have introduced an Embedding Layer as well which converts the sequence embeddings into unique vectors of equal size before feeding them to the Bi-LSTM layer.

```
def __bi_lstm_with_attention(self):
    inp = Input(shape=(self.in_size,))
    x = Embedding(self.vocab_size, self.embedding_size)(inp)
    x = Bidirectional(LSTM(self.vocab_size + 300,
                           return_sequences=True,
                           dropout=0.25,
                           recurrent_dropout=0.25))(x)
    x = Attention(self.in_size)(x)
    x = Dense(self.vocab_size + 100, activation="relu")(x)
    x = Dropout(0.25)(x)
    x = Dense(self.out_size, activation='softmax')(x)
    self.model = Model(inputs=inp, outputs=x)
```

```
    self.model.compile(loss='categorical_crossentropy',
                       optimizer='adam',
                       metrics=['accuracy'])
```

Step 14: Now, when everything is set we split the dataset into training and testing pairs and train and predict on them.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, random_state=42)
```

We have added to callbacks to the fit function for early stopping and model check point.

```
ef train(self):
    file_path = "./TableData/model.hdf5"
    ckpt = ModelCheckpoint(file_path,
                           monitor='val_loss',
                           verbose=1,
                           save_best_only=True,
                           mode='min')
    early = EarlyStopping(monitor="val_loss",
                          mode="min",
                          patience=1)
    self.model.fit(self.X_Train,
                   self.y_train,
                   verbose=1,
                   batch_size=10,
                   epochs=15,
                   validation_split=0.1,
                   callbacks=[ckpt, early])
    self.model.save_weights(file_path)
```

Risks and Limitations

Theoretically, this approach looks good and formidable, but due to lack of computation resources, the training is taking a lot of time and hence this solution demands a distributed tensor flow framework on few GPUs. Until that is achieved the actual efficacy of the approach remains unknown.