In [1]:

```
import sys
path = '/tf/notebooks/Capstone/backend/src'
sys.path.append(path)

import ktrain
from ktrain import text
from time_series_analysis.time_series import TimeSeries
import pandas as pd
import random
import spacy
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from tensorflow.keras.utils import to_categorical
import numpy as np
import math
```

In [2]:

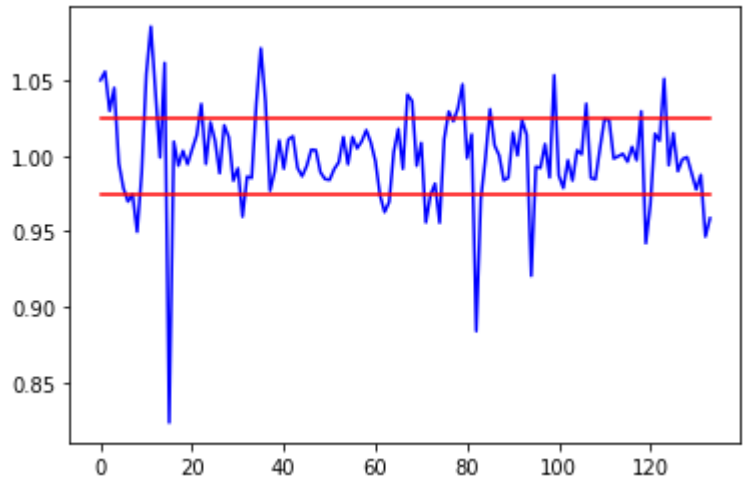
```
input_csv_text = '../../data/airliner_completed.csv'
csv = pd.read_csv(input_csv_text)
texts = csv[['Text', 'Datum']].values

for text_id in range(len(texts)):
    date = texts[text_id][1]
    if len(date) == 10:
        date = date[3:]
    if len(date) == 9:
        date = date[2:]
    if date == 'Datum':
        date = '00.0000'
    texts[text_id][1] = date

time_series = TimeSeries()
time_series.plot_results(time_series.get_residuums(), spread=0.025)
labels = time_series.get_residuums_dates(spread=0.025, four_cat=False)

def merge_data(texts, labels, sliding_window=1):
    final_texts = []
    final_labels = []
    for text_id in range(len(texts)):
        for label_id in range(len(labels) - sliding_window):
            if texts[text_id][1] == labels[label_id + sliding_window][1][3:]:
                final_texts.append(texts[text_id][0])
                final_labels.append(labels[label_id + sliding_window][0])
    return final_texts, final_labels

texts, labels = merge_data(texts, labels, sliding_window=4)
labels = to_categorical(labels)
print(len(texts))
```




```
30500
```

In [24]:

```
four_classes = False
epochs = 5
learning_rate = 5e-5
batch_size = 4
max_length = 512
max_words = 25000
```

In [4]:

```
nlp = spacy.load('de_core_news_sm')
stemmer = PorterStemmer()
stoplist = stopwords.words('german')
```

In [5]:

```
if four_classes:
    class_names = ['least', 'less', 'more', 'most']
else:
    class_names = ['less', 'equal', 'more']
```

In [6]:

```
def lemmatize_remove_stop(texts, stoplist):
    lemmatized_texts = []
    for document in list(nlp.pipe(texts, disable=['tagger', 'parser', 'ner'])):
        current_text = []
        for token in document:
            if token.lemma_ not in stoplist:
                current_text.append(token.lemma_)

        lemmatized_texts.append(' '.join(current_text))
    return lemmatized_texts

texts = lemmatize_remove_stop(texts, stoplist)
```

In [7]:

```
data = []
if four_classes:
    for t, label in zip(texts, labels):
        data.append([t, label[0], label[1], label[2], label[3]])
else:
    for t, label in zip(texts, labels):
        data.append([t, label[0], label[1], label[2]])
```

In [8]:

```
def split_test_data(data, split=0.1, random_seed=42):
    np.random.seed(random_seed)
    np.random.shuffle(data)
    split_item = math.floor(split * len(data))
    print('split at: ', split_item)
    x_test, y_test = data[:split_item, 0], data[:split_item, 1:]
    x_train, y_train = data[split_item:, 0], data[split_item:, 1:]
    return x_train, y_train, x_test, y_test
```

In [9]:

```
x_train, y_train, x_val, y_val = split_test_data(np.array(data), split=0.15, ran
dom_seed=4242)
print(len(x_train), len(y_train), len(x_val), len(y_val))
y_train = [[int(float(e)) for e in l] for l in y_train]
y_val = [[int(float(e)) for e in l] for l in y_val]
```

```
split at:  4575
25925 25925 4575 4575
```

In [10]:

```
from sklearn.utils import class_weight
def generate_balanced_weights(y_train):
    y_labels = [y.argmax() for y in np.array(y_train)]
    class_weights = class_weight.compute_class_weight('balanced', np.unique(y_la
bels), y_labels)
    weight_dict = {}
    for key in range(len(class_weights)):
        weight_dict[key] = class_weights[key]
    return weight_dict

class_weight_dict = generate_balanced_weights(y_train)
print(class_weight_dict)
```

```
{0: 2.574990067540723, 1: 0.4492911857474611, 2: 2.5912043978010995}
```

In [11]:

```
MODEL ='distilbert-base-multilingual-cased'
MODEL_bert = 'bert-base-german-cased'
transformer = text.Transformer(MODEL_bert, maxlen=max_length, class_names=class_
names)
train_data = transformer.preprocess_train(x_train, y_train)
val_data = transformer.preprocess_test(x_val, y_val)
```

```
preprocessing train...
language: de
train sequence lengths:
        mean : 201
        95percentile : 525
        99percentile : 890


Is Multi-Label? False
preprocessing test...
language: de
test sequence lengths:
        mean : 201
        95percentile : 524
        99percentile : 827
```

In [12]:

```
model = transformer.get_classifier()
```

In [25]:

```
learner = ktrain.get_learner(model, train_data=train_data, val_data=val_data, ba
tch_size=batch_size)
```

In [26]:

```
learner.fit_onecycle(5e-5, epochs=epochs, class_weight=class_weight_dict)
```

```
begin training using onecycle policy with max lr of 5e-05...
Train for 6482 steps, validate for 143 steps
Epoch 1/5
6482/6482 [==============================] - 2208s 341ms/step - los
s: 1.1074 - accuracy: 0.3453 - val_loss: 1.1073 - val_accuracy: 0.12
20
Epoch 2/5
6482/6482 [==============================] - 2221s 343ms/step - los
s: 1.1139 - accuracy: 0.3126 - val_loss: 1.1045 - val_accuracy: 0.12
20
Epoch 3/5
  82/6482 [..............................] - ETA: 35:14 - loss: 1.11
58 - accuracy: 0.3333
```

```
KeyboardInterruptTraceback (most recent call last)
<ipython-input-26-751fcf2885cd> in <module>
----> 1 learner.fit_onecycle(5e-5, epochs=epochs, class_weight=class
_weight_dict)

/usr/local/lib/python3.6/dist-packages/ktrain/core.py in fit_onecycl
e(self, lr, epochs, checkpoint_folder, cycle_momentum, max_momentum,
min_momentum, verbose, class_weight, callbacks)
    834         hist = self.fit(lr, epochs, early_stopping=None,
    835                         checkpoint_folder=checkpoint_folder,
--> 836                         verbose=verbose, class_weight=class_
weight, callbacks=kcallbacks)
    837         hist.history['lr'] = clr.history['lr']
    838         hist.history['iterations'] = clr.history['iteration
s']

/usr/local/lib/python3.6/dist-packages/ktrain/core.py in fit(self, l
r, n_cycles, cycle_len, cycle_mult, lr_decay, checkpoint_folder, ear
ly_stopping, class_weight, callbacks, verbose)
   1289                                           shuffle=True,
   1290                                           class_weight=class_w
eight,
-> 1291                                           callbacks=kcallback
s)
   1292         if sgdr is not None: hist.history['lr'] = sgdr.histo
ry['lr']
   1293         self.history = hist

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/keras/
engine/training.py in fit(self, x, y, batch_size, epochs, verbose, c
allbacks, validation_split, validation_data, shuffle, class_weight,
 sample_weight, initial_epoch, steps_per_epoch, validation_steps, va
lidation_freq, max_queue_size, workers, use_multiprocessing, **kwarg
s)
    817         max_queue_size=max_queue_size,
    818         workers=workers,
--> 819         use_multiprocessing=use_multiprocessing)
    820
    821   def evaluate(self,

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/keras/
engine/training_v2.py in fit(self, model, x, y, batch_size, epochs,
 verbose, callbacks, validation_split, validation_data, shuffle, cla
ss_weight, sample_weight, initial_epoch, steps_per_epoch, validation
_steps, validation_freq, max_queue_size, workers, use_multiprocessin
g, **kwargs)
    340                 mode=ModeKeys.TRAIN,
    341                 training_context=training_context,
--> 342                 total_epochs=epochs)
    343             cbks.make_logs(model, epoch_logs, training_resul
t, ModeKeys.TRAIN)
    344

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/keras/
engine/training_v2.py in run_one_epoch(model, iterator, execution_fu
nction, dataset_size, batch_size, strategy, steps_per_epoch, num_sam
ples, mode, training_context, total_epochs)
    126         step=step, mode=mode, size=current_batch_size) as ba
tch_logs:
    127       try:
```

```
--> 128         batch_outs = execution_function(iterator)
    129       except (StopIteration, errors.OutOfRangeError):
    130         # TODO(kaftan): File bug about tf function and error
s.OutOfRangeError?

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/keras/
engine/training_v2_utils.py in execution_function(input_fn)
     96     # `numpy` translates Tensors to values in Eager mode.
     97     return nest.map_structure(_non_none_constant_value,
---> 98                               distributed_function(input_f
n))
     99
    100   return execution_function

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/
def_function.py in __call__(self, *args, **kwds)
    566         xla_context.Exit()
    567     else:
--> 568       result = self._call(*args, **kwds)
    569
    570     if tracing_count == self._get_tracing_count():

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/
def_function.py in _call(self, *args, **kwds)
    597       # In this case we have created variables on the first
 call, so we run the
    598       # defunned version which is guaranteed to never create
variables.
--> 599       return self._stateless_fn(*args, **kwds)  # pylint: di
sable=not-callable
    600     elif self._stateful_fn is not None:
    601       # Release the lock early so that multiple threads can
 perform the call

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/
function.py in __call__(self, *args, **kwargs)
   2361     with self._lock:
   2362       graph_function, args, kwargs = self._maybe_define_func
tion(args, kwargs)
-> 2363     return graph_function._filtered_call(args, kwargs)  # py
lint: disable=protected-access
   2364
   2365   @property

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/
function.py in _filtered_call(self, args, kwargs)
   1609         if isinstance(t, (ops.Tensor,
   1610                           resource_variable_ops.BaseResourc
eVariable))),
-> 1611         self.captured_inputs)
   1612
   1613   def _call_flat(self, args, captured_inputs, cancellation_m
anager=None):

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/
function.py in _call_flat(self, args, captured_inputs, cancellation_
manager)
   1690       # No tape is watching; skip to running the function.
   1691       return self._build_call_outputs(self._inference_functi
on.call(
-> 1692           ctx, args, cancellation_manager=cancellation_manag
```

```
er))
   1693     forward_backward = self._select_forward_and_backward_fun
ctions(
   1694         args,

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/
function.py in call(self, ctx, args, cancellation_manager)
    543               inputs=args,
    544               attrs=("executor_type", executor_type, "config
_proto", config),
--> 545               ctx=ctx)
    546         else:
    547           outputs = execute.execute_with_cancellation(

/usr/local/lib/python3.6/dist-packages/tensorflow_core/python/eager/
execute.py in quick_execute(op_name, num_outputs, inputs, attrs, ct
x, name)
     59     tensors = pywrap_tensorflow.TFE_Py_Execute(ctx._handle,
 device_name,
     60                                                op_name, inpu
ts, attrs,
---> 61                                                num_outputs)
     62   except core._NotOkStatusException as e:
     63     if name is not None:

KeyboardInterrupt:
```

In [ ]:

```
predictor = ktrain.get_predictor(learner.model, preproc=transformer)
```

In [ ]:

```
confusion = learner.evaluate()
```

In [ ]:

```
# print confusion matrix
import matplotlib.pyplot as plt
import seaborn as sn
labels = class_names
cm_df = pd.DataFrame(confusion, labels, labels)
sn.set(font_scale=1.1, font='Arial')
ax = sn.heatmap(cm_df, cmap="Blues", annot=True, annot_kws={"size": 11}, cbar=Fa
lse, fmt='g')
ax.set_xlabel("Actual")
ax.set_ylabel("Predicted")
ax.set_title("Confusion Matrix")
plt.show()
```

In [ ]:

```
confusion = learner.evaluate()
```

In [ ]:

```

```