Download Dataset

In [ ]:
```
%%bash
if [ ! -d '/content/data' ]; then
  git clone https://gitlab.com/dimu1020/coms4995_fp.git '/content/data'
  cd '/content/data'
else
  echo "Dataset already downloaded in '/content/data'"
fi
```

```
Cloning into '/content/data'...
```

Global Dependencies

In [ ]:
```
import os
import pandas as pd
import numpy as np
import re
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import nltk
import math
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

Data Preprocessing

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
train_dat = pd.read_csv('data/train.csv')
X = train_dat['question_text']
y = train_dat['target']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=42)

len_X_train = np.array([len(sentence) for sentence in X_train])
len_X_positive = [i for (i,v) in zip(len_X_train,y_train) if v]
len_X_negative = [i for (i,v) in zip(len_X_train,y_train) if not v]
```

Word Embedding

In [ ]:
```
def text_cleaner(text):
    rules = [
        {r'>\s+': u''},  # remove spaces after a tag opens or closes
        {r'\s+': u' '},  # replace consecutive spaces
        {r'^\s+': u''}  # remove spaces at the beginning
    ]
    for rule in rules:
      for (k, v) in rule.items():
          regex = re.compile(k)
          text = regex.sub(v, text)
    text = text.rstrip()
    return text.lower()
```

In [ ]:
```
# %pip install autocorrect
nltk.download('punkt')
nltk.download('wordnet')
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
import string

if not os.path.exists("/content/data/cleaned_X.pkl"):
    cleaned_X = []
    stop_words = set(stopwords.words('english'))
    exclude = set(string.punctuation)
    lemmatizer = WordNetLemmatizer()
    porter = PorterStemmer()
    for row in X.tolist():
      cleaned_data = text_cleaner(row)
      words = word_tokenize(cleaned_data)
      wordsFiltered = []
      for w in words:
        if w not in stop_words and w not in exclude and not w.isdigit():
          w = porter.stem(w)
          w = lemmatizer.lemmatize(w)
          wordsFiltered.append(w)
      cleaned_X.append(wordsFiltered)
    cleaned_X = pd.Series(cleaned_X, name = 'question').to_frame()
    cleaned_X.to_pickle('/content/data/cleaned_X.pkl')
else:
    cleaned_X = pd.read_pickle("/content/data/cleaned_X.pkl")
cleaned_X.head()
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[ ]:

	question
0	[quebec, nationalist, see, provinc, nation, 1960]
1	[adopt, dog, would, encourag, peopl, adopt, shop]
2	[veloc, affect, time, veloc, affect, space, ge...
3	[otto, von, guerick, use, magdeburg, hemispher]
4	[convert, montra, helicon, mountain, bike, cha...

In [ ]:
```
data = cleaned_X.join(y)
data
```

Out[ ]:

	question	target
0	[quebec, nationalist, see, provinc, nation, 1960]	0
1	[adopt, dog, would, encourag, peopl, adopt, shop]	0
2	[veloc, affect, time, veloc, affect, space, ge...	0
3	[otto, von, guerick, use, magdeburg, hemispher]	0
4	[convert, montra, helicon, mountain, bike, cha...	0
...	...	...
1306117	[technic, skill, need, comput, scienc, undergr...	0
1306118	[m, ece, good, job, prospect, usa, like, india...	0
1306119	[foam, insul, toxic]	0
1306120	[one, start, research, project, base, biochemi...	0
1306121	[win, battl, wolverin, puma]	0

1306122 rows × 2 columns

Word2Vec

Word2Vec is a statistical method for efficiently learning a standalone word embedding from a text corpus. It was developed by Tomas Mikolov, et al. at Google in 2013 as a response to make the neural-network-based training of the embedding more efficient and since then has become the de facto standard for developing pre-trained word embedding.

In [ ]:
```
def get_w2v_model(data):
      '''
      Reference: https://www.kaggle.com/pierremegret/gensim-word2vec-tutorial#
Training-the-model
      '''
      from time import time  # To time our operations
      # initialize the parameters
      w2v_model = Word2Vec(min_count=20,
                           window=2,
                           size=300,
                           sample=6e-5,
                           alpha=0.03,
                           min_alpha=0.0007,
                           negative=20)
      # build vocabulary table
      t = time()
      w2v_model.build_vocab(data, progress_per=10000)
      print('Time to build vocab: {} mins'.format(round((time() - t) / 60, 2
)))

      # train the model
      t = time()
      w2v_model.train(data, total_examples=w2v_model.corpus_count, epochs=30,
report_delay=1)
      print('Time to train the model: {} mins'.format(round((time() - t) / 60,
2)))
      w2v_model.save("/content/data/w2v.model")
      w2v_model.init_sims(replace=True)
      return w2v_model
```

In [ ]:
```
from gensim.models import Word2Vec
if not os.path.exists("/content/data/w2v.model"):
    w2v_model = get_w2v_model(data['question'])
    w2v_model.save("/content/data/w2v.model")
else:
    w2v_model = Word2Vec.load("/content/data/w2v.model")
```

```
Time to build vocab: 0.15 mins
Time to train the model: 16.8 mins
```

In [ ]:
```
# small test
w2v_model.most_similar('hello')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWa
rning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, us
e self.wv.most_similar() instead).
```

Out[ ]:
```
[('hi', 0.502326488494873),
 ('hey', 0.3973042368888855),
 ('goodby', 0.3699946701526642),
 ('talk', 0.3574012219905853),
 ('messag', 0.3557403087615967),
 ('say', 0.35483986139297485),
 ('greet', 0.352449506521225),
 ('dear', 0.34061509370803833),
 ('yeah', 0.33764439821243286),
 ('repli', 0.3323920965194702)]
```

In [ ]:
```
word_list = list(w2v_model.wv.vocab)
vectors = w2v_model.wv[word_list]
```

In [ ]:
```
if not os.path.exists("/content/data/df_final.pkl"):
    # get each word vector of each title and sum up to get sentence vector
    def word_to_sentence_vec(l,model):
        l = ' '.join(['quebec', 'nationalist', 'see', 'provinc', 'nation', '19
60'])
        words = l.split(' ')
        ret = np.empty((1, 300), dtype=np.float32)
        for w in words:
            if w in model.wv:
                ret += model.wv.__getitem__([w])
        return ret[0]

    def get_vectorized_title_df(data, model):
        df_vectorized = []
        df_vectorized.append(data.map(lambda x: word_to_sentence_vec(x, model
)))
        return df_vectorized

    li = get_vectorized_title_df(data['question'], w2v_model)
    df_final = data['target'].to_frame()
    df_final['question'] = pd.concat(li, axis=1)
    df_final = df_final[['question', 'target']]
    df_final.to_pickle('/content/data/df_final.pkl')
else:
    df_final = pd.read_pickle("/content/data/df_final.pkl")
df_final
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: RuntimeWarnin
g: invalid value encountered in add
  if __name__ == '__main__':
```

Out[ ]:

	question	target
0	[3.300651e+26, -0.0023040287, 0.2824797, -0.10...	0
1	[3.3015953e+26, -0.0023040287, 0.2579247, -0.0...	0
2	[3.3020675e+26, -0.0023040287, 0.2579247, -0.0...	0
3	[3.3025398e+26, -0.0023040287, 0.2579247, -0.0...	0
4	[-0.07357857, -0.0023040287, 0.2579247, -0.020...	0
...	...	...
1306117	[-9.519934e+19, -0.0023040287, 0.2579247, -0.0...	0
1306118	[-9.52106e+19, -0.0023040287, 0.2579247, -0.02...	0
1306119	[-9.522186e+19, -0.0023040287, 0.2579247, -0.0...	0
1306120	[-9.523312e+19, -0.0023040287, 0.2579247, -0.0...	0
1306121	[-9.524438e+19, -0.0023040287, 0.2579247, -0.0...	0

1306122 rows × 2 columns

Exploratory Data Analysis and Visualization

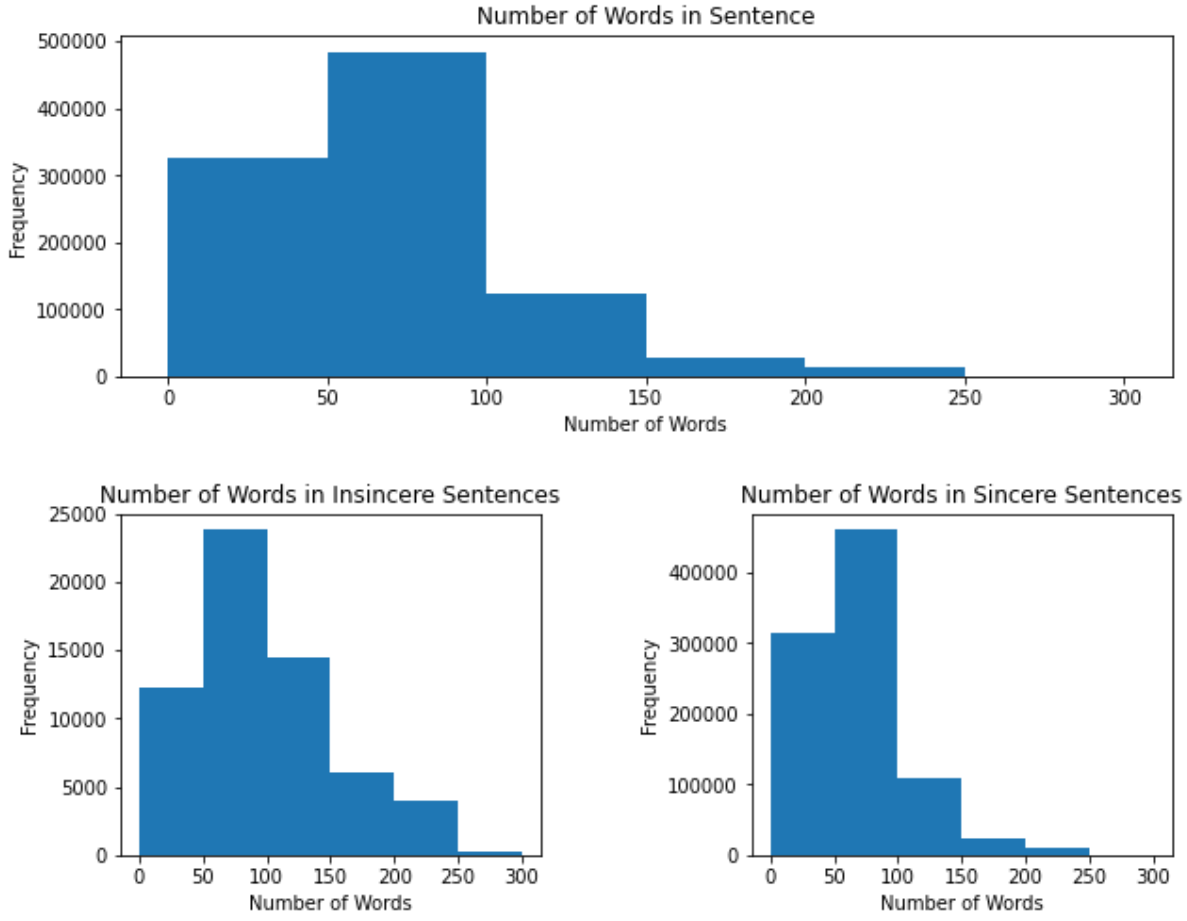
Distribution

In [ ]:
```
#censored words analysis
train_dat['star'] = train_dat.apply(lambda row: "**" in row.question_text, axis=1)
star_text = train_dat[train_dat['star']==True]
print(np.mean(star_text['target']))
print(np.mean(train_dat['target']))
```

```
0.5789473684210527
0.06187017751787352
```

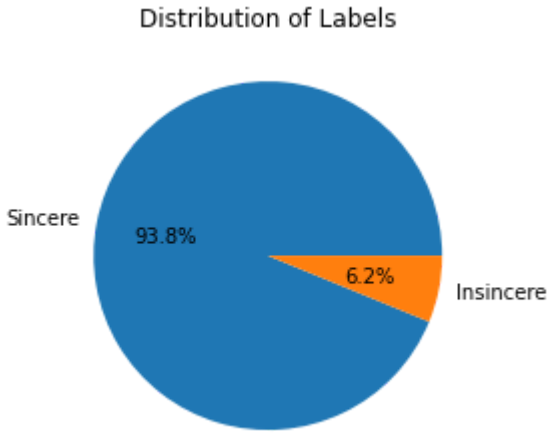
Hist Plot

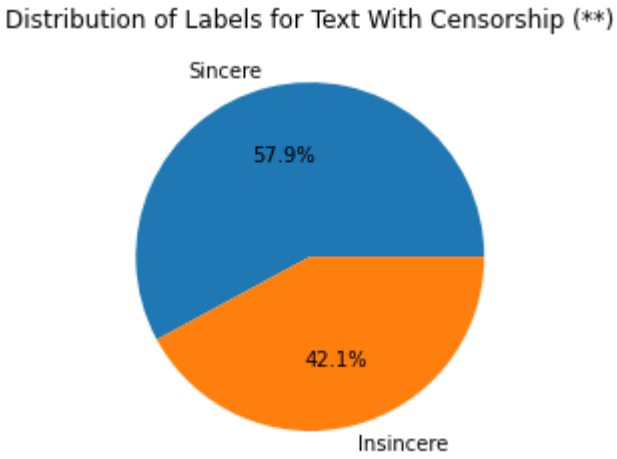
In [ ]:
```
fig = plt.figure(figsize=(10,8))
top = fig.add_subplot(211)
bottom_left = fig.add_subplot(223)
bottom_right = fig.add_subplot(224)

top.hist(len_X_train, bins=np.linspace(0, 300, 7))
bottom_left.hist(len_X_positive,bins = np.linspace(0,300,7),label='insincere')
bottom_right.hist(len_X_negative,bins = np.linspace(0,300,7),label='sincere')

for plot in (top, bottom_left, bottom_right):
    plot.set_xlabel("Number of Words")
    plot.set_ylabel("Frequency")

plt.subplots_adjust(wspace=0.5,hspace=0.4)

top.set_title("Number of Words in Sentence")
bottom_left.set_title("Number of Words in Insincere Sentences")
bottom_right.set_title("Number of Words in Sincere Sentences")
plt.show()
```



Pie Chart

In [ ]:
```
values = train_dat['target'].value_counts()
plt.pie(values, labels=["Sincere", "Insincere"], autopct='%1.1f%%')
plt.title("Distribution of Labels")
plt.show()
values = star_text['target'].value_counts()
plt.pie(values, labels=["Sincere", "Insincere"], autopct='%1.1f%%')
plt.title("Distribution of Labels for Text With Censorship (**)")
plt.show()
```





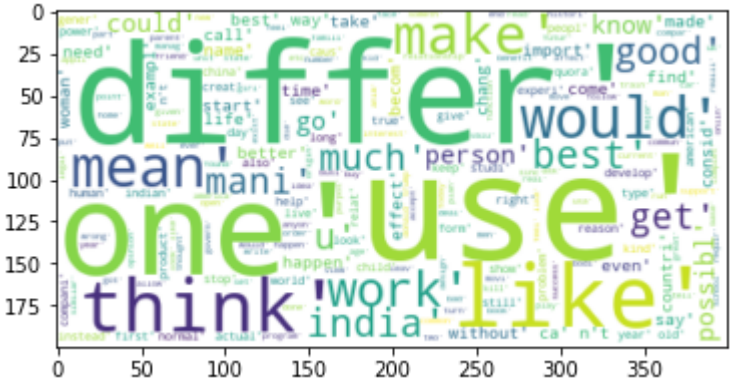
Word Cloud

In [ ]:
```
cleaned_X_train, cleaned_X_test, y_train, y_test = train_test_split(cleaned_X, y)
```

In [ ]:
```
import numpy as np
import pandas as pd
from os import path
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
clean_X_sincere = [i for (i,v) in zip(cleaned_X_train,y_train) if not v]
clean_X_insincere = [i for (i,v) in zip(cleaned_X_train,y_train) if v]

def draw_word_cloud(data):
  temp = []
  for dat in data:
    while("n't" in dat):
      dat.remove("n't")
      temp.append(' '.join(dat))
      f = ' '.join(temp)
      wordcloud = WordCloud().generate(f)
      plt.imshow(wordcloud, interpolation='bilinear')
      plt.axis("off")
      plt.show()
draw_word_cloud(clean_X_sincere)
draw_word_cloud(clean_X_insincere)
```

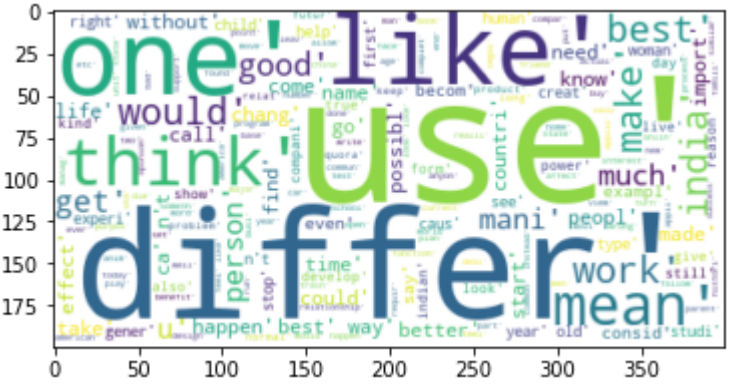
In [ ]:
```
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
# Display the generated image
data_0 = data[data['target'] == 0]
data_1 = data[data['target'] == 1]
text = []
for item in data['question'].values:
  for i in item:
    text.append(i)

wordcloud = WordCloud(background_color='white', max_words=1000).generate(str(t
ext))

plt.imshow(wordcloud, interpolation="bilinear")
plt.figure(figsize=[8,10])
plt.axis("off")
```

Out[ ]: (0.0, 1.0, 0.0, 1.0)



In [ ]:
```
text = []
for item in data_0['question'].values:
  for i in item:
    text.append(i)

wordcloud = WordCloud(background_color='white', max_words=1000).generate(str(t
ext))

plt.imshow(wordcloud, interpolation="bilinear")
plt.figure(figsize=[8,10])
plt.axis("off")
```

Out[ ]: (0.0, 1.0, 0.0, 1.0)



In [ ]:
```
text = []
for item in data_1['question'].values:
  for i in item:
    text.append(i)


wordcloud = WordCloud(background_color='white', max_words=1000).generate(str(t
ext))

plt.imshow(wordcloud, interpolation="bilinear")
plt.figure(figsize=[8,10])
plt.axis("off")
```

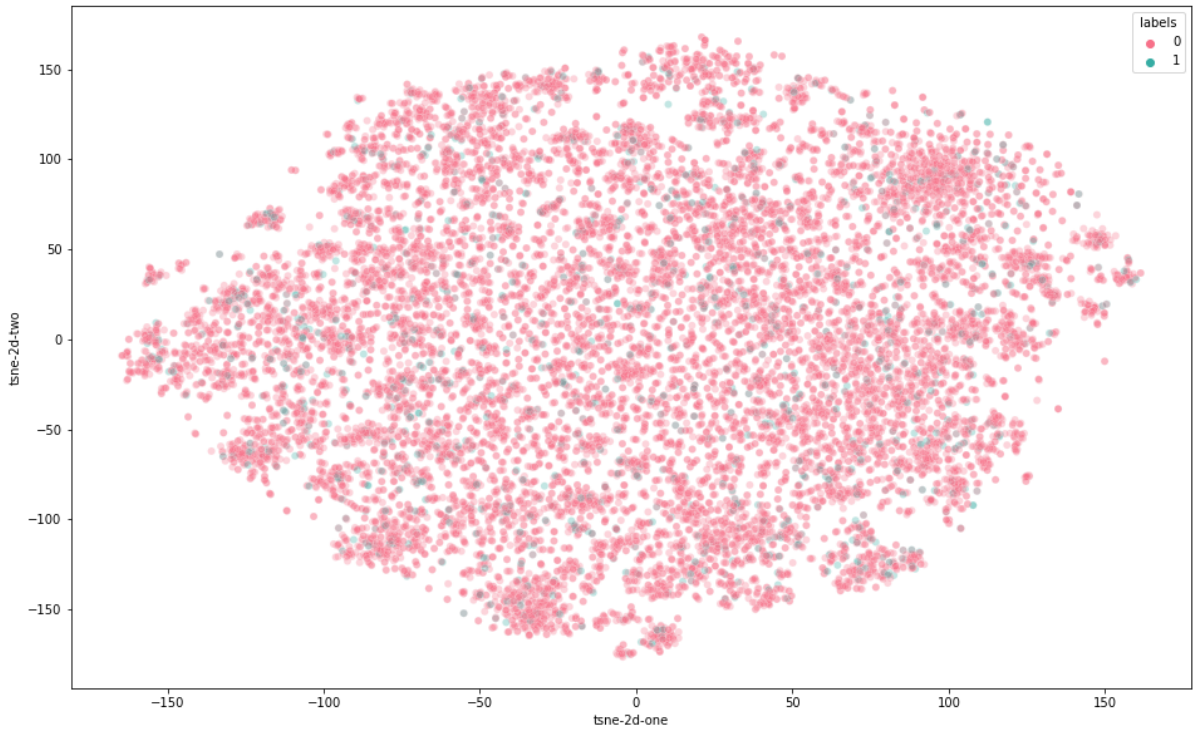
Out[ ]: (0.0, 1.0, 0.0, 1.0)

tsne plot

In [ ]:
```
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, verbose=1, perplexity=10, n_iter=3000,init='pca',r
andom_state=42)
tsne_results = tsne.fit_transform(vectors)
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
merged_frame_rolled_up = pd.DataFrame()
merged_frame_rolled_up['tsne-2d-one'] = tsne_results[:,0]
merged_frame_rolled_up['tsne-2d-two'] = tsne_results[:,1]
merged_frame_rolled_up['labels'] = y
plt.figure(figsize=(16,10))
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue = 'labels',
    palette=sns.color_palette("husl", len(merged_frame_rolled_up['labels'].uni
que())),
    data=merged_frame_rolled_up,
    legend="full",
    alpha=0.3
)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: Future
Warning: The default learning rate in TSNE will change from 200.0 to 'auto' i
n 1.2.
  FutureWarning,

[t-SNE] Computing 31 nearest neighbors...
[t-SNE] Indexed 16602 samples in 0.004s...
[t-SNE] Computed neighbors for 16602 samples in 11.234s...
[t-SNE] Computed conditional probabilities for sample 1000 / 16602
[t-SNE] Computed conditional probabilities for sample 2000 / 16602
[t-SNE] Computed conditional probabilities for sample 3000 / 16602
[t-SNE] Computed conditional probabilities for sample 4000 / 16602
[t-SNE] Computed conditional probabilities for sample 5000 / 16602
[t-SNE] Computed conditional probabilities for sample 6000 / 16602
[t-SNE] Computed conditional probabilities for sample 7000 / 16602
[t-SNE] Computed conditional probabilities for sample 8000 / 16602
[t-SNE] Computed conditional probabilities for sample 9000 / 16602
[t-SNE] Computed conditional probabilities for sample 10000 / 16602
[t-SNE] Computed conditional probabilities for sample 11000 / 16602
[t-SNE] Computed conditional probabilities for sample 12000 / 16602
[t-SNE] Computed conditional probabilities for sample 13000 / 16602
[t-SNE] Computed conditional probabilities for sample 14000 / 16602
[t-SNE] Computed conditional probabilities for sample 15000 / 16602
[t-SNE] Computed conditional probabilities for sample 16000 / 16602
[t-SNE] Computed conditional probabilities for sample 16602 / 16602
[t-SNE] Mean sigma: 0.232501

/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:986: Future
Warning: The PCA initialization in TSNE will change to have the standard devi
ation of PC1 equal to 1e-4 in 1.2. This will ensure better convergence.
  FutureWarning,

[t-SNE] KL divergence after 250 iterations with early exaggeration: 110.06106
6
[t-SNE] KL divergence after 3000 iterations: 2.501515
```

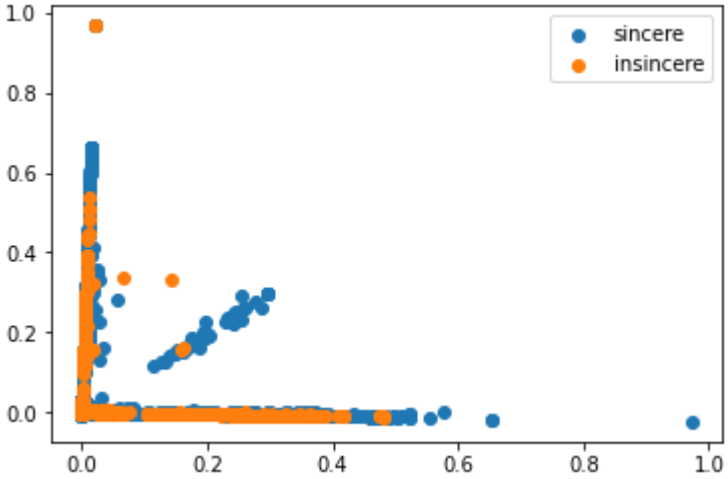
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f003a122410>



Modeling

In [ ]:
```
# %pip install transformer
# from transformers import BertModel, BertConfig, BertTokenizer
# tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
# model = BertModel.from_pretrained('bert-base-uncased')

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA,TruncatedSVD
cleaned_sentences = []
vec = TfidfVectorizer(ngram_range = (2,2))
for i in range(len(cleaned_X['question'])):
  cleaned_sentences.append(" ".join(cleaned_X['question'][i]))
vec_X_train = vec.fit_transform(cleaned_sentences)
svd = TruncatedSVD(n_components = 2)
vec_X_train = svd.fit_transform(vec_X_train)
vec_sincere = [i for (i,v) in zip(vec_X_train,y_train) if not v]
vec_insincere = [i for (i,v) in zip(vec_X_train,y_train) if v]
```

In [ ]:
```
plt.scatter([tmp[0] for tmp in vec_sincere],[tmp[1] for tmp in vec_sincere],la
bel = 'sincere')
plt.scatter([tmp[0] for tmp in vec_insincere],[tmp[1] for tmp in vec_insincere
],label ='insincere')
plt.legend()
plt.show()
```



In [ ]:
```
# get 50,000 positive and 50,000 negative samples
# make sure we have variables: cleaned_X, y

samples_per_class = 50000

neg = np.where(y == 0)
pos = np.where(y == 1)
neg = neg[0][:samples_per_class]
pos = pos[0][:samples_per_class]

X_neg_df = pd.DataFrame(cleaned_X["question"][neg])
X_pos_df = pd.DataFrame(cleaned_X["question"][pos])
new_X = pd.concat([X_neg_df, X_pos_df], ignore_index=True)
new_Y = [0] * samples_per_class + [1] * samples_per_class

# use new_X, new_Y
```

Logistic Regression

In [ ]:
```
#baseline logistic regression with bag-of-words
new_X['star'] = new_X.apply(lambda row: "**" in row.question, axis=1)

X_train, X_test, y_train, y_test = train_test_split(
    new_X, np.array(new_Y).reshape(-1), random_state=42, test_size=0.2)
```

In [ ]:
```
def to_corpus(X_train):
  i = 0
  text = []
  for row in X_train['question'].reset_index(drop = True):
    row = ' '.join(row)
    text.append(row)
  X_feed_train = pd.DataFrame(text, columns = ['question_text'])
  return X_feed_train

X_feed_train = to_corpus(X_train)
X_feed_test = to_corpus(X_test)
```

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer
vector = CountVectorizer(stop_words='english')

X_train_txt = vector.fit_transform(X_feed_train['question_text'])
X_test_txt = vector.transform(X_feed_test['question_text'])

#tried adding indicator of censored words as a feature, no improvement
from scipy.sparse import hstack
X_train_vec = hstack((X_train_txt,np.array(X_train['star']*1)[:,None]))
X_test_vec = hstack((X_test_txt,np.array(X_test['star']*1)[:,None]))

from sklearn.linear_model import LogisticRegressionCV
lr = LogisticRegressionCV(max_iter=1000,n_jobs=-1).fit(X_train_vec,y_train)
lr.score(X_test_vec,y_test)
```

Out[ ]:
```
0.87165
```

LSTM

In [ ]:
```
vocab_size = 10000
seq_length = 100
embedding_size = 128
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(new_X["question"])
```

In [ ]:
```
sequences = tokenizer.texts_to_sequences(new_X["question"])
```

In [ ]:
```
X_sequence = keras.preprocessing.sequence.pad_sequences(sequences, maxlen=seq_
length)
```

In [ ]:
```
from keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

X_train, X_test, y_train, y_test = train_test_split(np.array(X_sequence), np.a
rray(new_Y), test_size=0.2)
model = Sequential()

model.add(Embedding(vocab_size, embedding_size, input_length=seq_length))
model.add(LSTM(64))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
```

In [ ]:
```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 100, 128)          1280000

 lstm_2 (LSTM)               (None, 64)                49408

 dense_1 (Dense)             (None, 32)                2080

 dropout (Dropout)           (None, 32)                0

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 1,331,521
Trainable params: 1,331,521
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
from keras import callbacks
es = callbacks.EarlyStopping(monitor='val_loss', patience=4)

model.fit(X_train, y_train, epochs=100, batch_size=128, validation_split=0.2,
callbacks=[es])
```

```
Epoch 1/100
500/500 [==============================] - 24s 38ms/step - loss: 0.3689 - acc
uracy: 0.8447 - val_loss: 0.3118 - val_accuracy: 0.8704
Epoch 2/100
500/500 [==============================] - 18s 37ms/step - loss: 0.2751 - acc
uracy: 0.8940 - val_loss: 0.3122 - val_accuracy: 0.8744
Epoch 3/100
500/500 [==============================] - 18s 36ms/step - loss: 0.2384 - acc
uracy: 0.9071 - val_loss: 0.3313 - val_accuracy: 0.8724
Epoch 4/100
500/500 [==============================] - 18s 35ms/step - loss: 0.2011 - acc
uracy: 0.9197 - val_loss: 0.3741 - val_accuracy: 0.8679
Epoch 5/100
500/500 [==============================] - 18s 35ms/step - loss: 0.1747 - acc
uracy: 0.9293 - val_loss: 0.4239 - val_accuracy: 0.8670
```

Out[ ]:
```
<keras.callbacks.History at 0x7f590f6e8990>
```

In [ ]:
```
eval = model.evaluate(X_test, y_test, batch_size=128)
```

```
157/157 [==============================] - 2s 10ms/step - loss: 0.4074 - accu
racy: 0.8708
```

Bert

In [ ]:
```
pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.7/dist-
packages (4.13.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.7/dist-p
ackages (from transformers) (6.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/di
st-packages (from transformers) (21.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-pack
ages (from transformers) (3.4.0)
Requirement already satisfied: tokenizers<0.11,>=0.10.1 in /usr/local/lib/pyt
hon3.7/dist-packages (from transformers) (0.10.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-p
ackages (from transformers) (1.19.5)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/
dist-packages (from transformers) (2019.12.20)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-pa
ckages (from transformers) (4.62.3)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.
7/dist-packages (from transformers) (4.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-pack
ages (from transformers) (2.23.0)
Requirement already satisfied: sacremoses in /usr/local/lib/python3.7/dist-pa
ckages (from transformers) (0.0.46)
Requirement already satisfied: huggingface-hub<1.0,>=0.1.0 in /usr/local/lib/
python3.7/dist-packages (from transformers) (0.2.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/p
ython3.7/dist-packages (from huggingface-hub<1.0,>=0.1.0->transformers) (3.1
0.0.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/pyt
hon3.7/dist-packages (from packaging>=20.0->transformers) (3.0.6)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-pac
kages (from importlib-metadata->transformers) (3.6.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /us
r/local/lib/python3.7/dist-packages (from requests->transformers) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
7/dist-packages (from requests->transformers) (2021.10.8)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests->transformers) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/
dist-packages (from requests->transformers) (3.0.4)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages
(from sacremoses->transformers) (1.15.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packag
es (from sacremoses->transformers) (1.1.0)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-package
s (from sacremoses->transformers) (7.1.2)
```

In [ ]:
```
from transformers import BertModel, BertConfig, BertTokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when
initializing BertModel: ['cls.predictions.transform.dense.weight', 'cls.seq_r
elationship.weight', 'cls.predictions.decoder.weight', 'cls.predictions.trans
form.dense.bias', 'cls.predictions.transform.LayerNorm.bias', 'cls.prediction
s.bias', 'cls.seq_relationship.bias', 'cls.predictions.transform.LayerNorm.we
ight']
- This IS expected if you are initializing BertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing
a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint
of a model that you expect to be exactly identical (initializing a BertForSeq
uenceClassification model from a BertForSequenceClassification model).
```

In [ ]:
```
model = model.eval()
import time
now = time.time()
def bert_embedding(dataset):
  vec = []
  for i,sentence in enumerate(dataset):
    if(i%500 == 0):
      print(i)
      print(time.time() - now)
    with torch.no_grad():
        inputs = tokenizer(' '.join(sentence), padding=True, return_tensors="p
t")
        output = model(**inputs, output_hidden_states=True)
        vec.append(output.last_hidden_state[0].detach().numpy())
  return vec
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=42)
```

In [ ]:
```
X_train_bert = bert_embedding(X_train)
X_test_bert = bert_embedding(X_test)
```

```
0
4.66516637802124
500
141.91749739646912
1000
263.95116662979126
1500
391.2345521450043
2000
522.2976984977722
2500
644.932214975357
3000
765.0675778388977
3500
886.3231303691864
4000
1013.7281243801117
4500
1137.83806681633
5000
1255.0060617923737
5500
1373.818082332611
6000
1500.5209090709686
6500
1627.2145426273346
7000
1746.2844426631927
7500
1869.9930906295776
8000
1992.3085839748383
8500
2114.05184674263
9000
2237.4052064418793
9500
2356.977357149124
10000
2475.021162748337
10500
2596.8844754695892
11000
2724.1991069316864
11500
2847.1816358566284
12000
2972.5426902770996
12500
3099.863466024399
13000
3214.5542283058167
13500
3339.8003249168396
14000
```

```
3463.61439371109
14500
3589.4159212112427
15000
3714.4412517547607
15500
3828.0130355358124
16000
3947.352865934372
16500
4067.3570699691772
17000
4189.168010473251
17500
4311.3889100551605
18000
4433.078191995621
18500
4554.273327112198
19000
4672.158997535706
19500
4799.494338750839
20000
4926.969295978546
20500
5049.610688209534
21000
5172.0337455272675
21500
5295.134817838669
22000
5421.098764896393
22500
5542.674609661102
23000
5671.213079690933
23500
5796.133297920227
24000
5918.6123814582825
24500
6049.332072973251
25000
6174.821154117584
25500
6297.65788435936
26000
6414.834669589996
26500
6533.730060577393
27000
6655.013530015945
27500
6779.695266008377
28000
6901.315932035446
```

```
28500
7022.489265918732
29000
7145.4323399066925
29500
7267.311827421188
30000
7391.537642478943
30500
7523.519173145294
31000
7645.429723978043
31500
7763.542230844498
32000
7889.1371483802795
32500
8008.829852581024
33000
8127.951533317566
33500
8250.846829891205
34000
8373.89423584938
34500
8495.897976636887
35000
8620.865504026413
35500
8743.462728977203
36000
8865.986606121063
36500
8986.784987211227
37000
9107.414498567581
37500
9230.180854558945
38000
9355.044885873795
38500
9480.702283143997
39000
9603.285409927368
```

In [ ]:
```
class lstm_with_bert(nn.Module):
  def __init__(self,embedding_dim,hidden_dim):
    super(lstm_with_bert, self).__init__()
    self.lstm = nn.LSTM(embedding_dim, hidden_dim,batch_first=True)
    self.fc1 = nn.Linear(hidden_dim, 32)
    self.fc2 = nn.Linear(32,1)

  def forward(self,input):
    lstm_out, (hn,cn) = self.lstm(input)
    fc_out = self.fc1(hn[0])
    fc_out = nn.ReLU()(fc_out)
    fc_out = nn.Dropout(p=0.2)(fc_out)
    fc2_out = self.fc2(fc_out)
    output = nn.Sigmoid()(fc2_out)
    return output[0].float()
```

In [ ]:
```
EMBEDDING_DIM = 768
HIDDEN_DIM = 128
model = lstm_with_bert(EMBEDDING_DIM, HIDDEN_DIM)
loss_function = nn.BCELoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)

for epoch in range(10):
    for i,(sentence, tag) in enumerate(zip(X_train_bert,y_train)):
        t_tot_loss = 0.0
        model.zero_grad()
        pred = model(torch.tensor(sentence).unsqueeze(dim=0))
        loss = loss_function(pred, torch.tensor(tag).float().unsqueeze(dim=0))
        loss.backward()
        optimizer.step()
        t_tot_loss += loss.cpu().item()
        if(i % 1000 == 0):
          print(i)
    print("epoch=%s t_loss=%0.4f" % (epoch, t_tot_loss))
```

In [ ]:
```
gold = []
predicted = []
with torch.no_grad():
    for X_b, y_b in zip(X_test_bert,y_test):
        y_pred = model(torch.tensor(X_b).unsqueeze(dim=0))
        gold.extend(torch.tensor(y_b).float().unsqueeze(dim=0))
        predicted.extend(y_pred)
predicted = np.around(predicted)
from sklearn.metrics import accuracy_score
accuracy_score(gold,predicted)
```

Transformer Model

In [ ]:
```
def take_first_fifty_thousand(df_final):
    slice_neg = df_final[df_final['target']==0][:50000].reset_index(drop = Tru
e)
    slice_pos = df_final[df_final['target']==1][:50000].reset_index(drop = Tru
e)
    df_final_slice = pd.concat([slice_neg, slice_pos])
    return df_final_slice
df_final_slice = take_first_fifty_thousand(df_final).reset_index(drop = True)
x_train, x_test, y_train, y_test = train_test_split(
    df_final_slice['question'].to_numpy(), df_final_slice['target'].to_numpy
(), random_state=42)
```

In [ ]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
class TransformerBlock(layers.Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super(TransformerBlock, self).__init__()
        self.att = layers.MultiHeadAttention(num_heads=num_heads, key_dim=embe
d_dim)
        self.ffn = keras.Sequential(
            [layers.Dense(ff_dim, activation="relu"), layers.Dense(embed_dim
),]
        )
        self.layernorm1 = layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = layers.LayerNormalization(epsilon=1e-6)
        self.dropout1 = layers.Dropout(rate)
        self.dropout2 = layers.Dropout(rate)

    def call(self, inputs, training):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)

class TokenAndPositionEmbedding(layers.Layer):
    def __init__(self, maxlen, vocab_size, embed_dim):
        super(TokenAndPositionEmbedding, self).__init__()
        self.token_emb = layers.Embedding(input_dim=vocab_size, output_dim=emb
ed_dim)
        self.pos_emb = layers.Embedding(input_dim=maxlen, output_dim=embed_dim
)

    def call(self, x):
        maxlen = tf.shape(x)[-1]
        positions = tf.range(start=0, limit=maxlen, delta=1)
        positions = self.pos_emb(positions)
        x = self.token_emb(x)
        return x + positions
```

In [ ]:
```
print(len(x_train), "Training sequences")
print(len(x_test), "Test sequences")
maxlen = 200
x_train = keras.preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = keras.preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

```
75000 Training sequences
25000 Test sequences
```

In [ ]:
```
embed_dim = 32  # Embedding size for each token
num_heads = 2  # Number of attention heads
ff_dim = 32  # Hidden layer size in feed forward network inside transformer
vocab_size = 20000 # size of the vocabulary

inputs = layers.Input(shape=(maxlen,))
embedding_layer = TokenAndPositionEmbedding(maxlen, vocab_size, embed_dim)
x = embedding_layer(inputs)
transformer_block = TransformerBlock(embed_dim, num_heads, ff_dim)
x = transformer_block(x)
x = layers.GlobalAveragePooling1D()(x)
x = layers.Dropout(0.1)(x)
x = layers.Dense(20, activation="relu")(x)
x = layers.Dropout(0.1)(x)
outputs = layers.Dense(2, activation="softmax")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

Training

In [ ]:
```
# model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metr
ics=["accuracy"])
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer=keras.optimizers.Adam(learning_rate=1e-4),
    metrics=["sparse_categorical_accuracy"],
)
model.summary()

callbacks = [keras.callbacks.EarlyStopping(patience=10, restore_best_weights=T
rue)]
history = model.fit(
    x_train, y_train,
    batch_size=32, epochs=10, validation_split=0.2,
    callbacks=callbacks,
)
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 200)]             0

 token_and_position_embeddin  (None, 200, 32)          646400
 g (TokenAndPositionEmbeddin
 g)

 transformer_block (Transfor  (None, 200, 32)          10656
 merBlock)

 global_average_pooling1d (G  (None, 32)               0
 lobalAveragePooling1D)

 dropout_2 (Dropout)         (None, 32)                0

 dense_2 (Dense)             (None, 20)                660

 dropout_3 (Dropout)         (None, 20)                0

 dense_3 (Dense)             (None, 2)                 42

=================================================================
Total params: 657,758
Trainable params: 657,758
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1875/1875 [==============================] - 267s 142ms/step - loss: 0.6941 -
sparse_categorical_accuracy: 0.4992 - val_loss: 0.6931 - val_sparse_categoric
al_accuracy: 0.5047
Epoch 2/10
1875/1875 [==============================] - 239s 128ms/step - loss: 0.6933 -
sparse_categorical_accuracy: 0.5016 - val_loss: 0.6931 - val_sparse_categoric
al_accuracy: 0.5047
Epoch 3/10
1875/1875 [==============================] - 243s 130ms/step - loss: 0.6932 -
sparse_categorical_accuracy: 0.4986 - val_loss: 0.6932 - val_sparse_categoric
al_accuracy: 0.4953
Epoch 4/10
1875/1875 [==============================] - 238s 127ms/step - loss: 0.6933 -
sparse_categorical_accuracy: 0.4976 - val_loss: 0.6931 - val_sparse_categoric
al_accuracy: 0.5047
Epoch 5/10
1875/1875 [==============================] - 241s 129ms/step - loss: 0.6932 -
sparse_categorical_accuracy: 0.5015 - val_loss: 0.6931 - val_sparse_categoric
al_accuracy: 0.5047
Epoch 6/10
1875/1875 [==============================] - 237s 127ms/step - loss: 0.6932 -
sparse_categorical_accuracy: 0.4949 - val_loss: 0.6931 - val_sparse_categoric
al_accuracy: 0.4953
Epoch 7/10
1875/1875 [==============================] - 241s 129ms/step - loss: 0.6932 -
sparse_categorical_accuracy: 0.5012 - val_loss: 0.6932 - val_sparse_categoric
al_accuracy: 0.4953
```

```
Epoch 8/10
1875/1875 [==============================] - 241s 128ms/step - loss: 0.6932 -
sparse_categorical_accuracy: 0.5024 - val_loss: 0.6932 - val_sparse_categoric
al_accuracy: 0.4953
Epoch 9/10
1875/1875 [==============================] - 238s 127ms/step - loss: 0.6932 -
sparse_categorical_accuracy: 0.5013 - val_loss: 0.6932 - val_sparse_categoric
al_accuracy: 0.4953
Epoch 10/10
1875/1875 [==============================] - 235s 125ms/step - loss: 0.6931 -
sparse_categorical_accuracy: 0.5006 - val_loss: 0.6932 - val_sparse_categoric
al_accuracy: 0.4953
```

Evaluation

In [ ]:
```
model.evaluate(x_test, y_test, verbose=1)
```

```
782/782 [==============================] - 36s 45ms/step - loss: 0.6932 - spa
rse_categorical_accuracy: 0.4989
```

Out[ ]:
```
[0.6931676268577576, 0.498879998922348]
```

Loss and Accuracy Plot

In [ ]:
```
sparse_categorical_accuracy = history.history['sparse_categorical_accuracy']
val_sparse_categorical_accuracy = history.history['val_sparse_categorical_accu
racy']
n_epochs = range(10)

plt.figure(figsize=(9, 7))
plt.plot(n_epochs, sparse_categorical_accuracy, 'r-', label='Train Accuracy',
color='darkblue')
if val_sparse_categorical_accuracy is not None:
    plt.plot(n_epochs, val_sparse_categorical_accuracy, 'r-', label='Test Accu
racy', color='brown')
plt.legend(loc=0)
plt.xlabel('Epoch')
plt.ylabel('Sparse Categorical Accuracy')
plt.title('Accuracy of the Training and Validation data For Transformer Model'
)
plt.show()
```

In [ ]:
```
loss = history.history.get('loss')
val_loss = history.history.get('val_loss')

n_epochs = range(10)

plt.figure(figsize=(9, 7))
plt.plot(n_epochs, loss, 'r-', label='Training loss', color='darkblue')
if val_loss is not None:
    plt.plot(n_epochs, val_loss, 'r-', label='Validation loss', color='brown')
plt.legend(loc=0)
plt.xlabel('Epoch')
plt.ylabel('Loss value')
plt.title('Loss of the Training and Validation data For Transformer Model')
plt.show()
```