Load data

```
# load text data and convert the label/sentiment into corresponding numeric values: '
# possible packages you might need are: pandas, numpy

import pandas as pd
import numpy as np

# read the training data


# get texts and labels


# show the first 5 records
df_train.head()
```

	text	sentiment	labels
0	Heres a single to add to Kindle. Just read t...	neutral	1
1	If you tire of Non-Fiction.. Check out http://...	neutral	1
2	Ghost of Round Island is supposedly nonfiction.	neutral	1
3	Why is Barnes and Nobles version of the Kindle...	negative	0
4	@Maria: Do you mean the Nook? Be careful bo...	positive	2

Preprocess dat

```
# preprocess the loaded textual data, including removing stopwords, stemming, and tok
# represent each document (i.e., comment) using TF-IDF strategy. The features are the
# possible packages you might need are: scikit-learn, numpy
from sklearn.feature_extraction.text import TfidfVectorizer


# tokenize and create a document-feature matrix X and a label vector Y




# print out the shape of X and Y
print(X.shape,',',Y.shape)
```

```
(1999, 500) , (1999,)
```

Traditional Machine Learning Models: Random Forest

```
# using 10-fold cross-validation to show the prediction accuracy
# possible packages you might need are: scikit-learn, numpy

from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier



print("Random Forest - mean: %.4f%% (std: +/- %.4f%%)" % (np.mean(rf_cvscores)*100, n
```

```
Random Forest - mean: 64.1332% (std: +/- 2.0919%)
```

Fully connected feedforward Neural Network

```
# Design your own network with the following requirements:
# 1. Having dropout
# 2. Separate the dataset into training and validation (80-20%)
# 3. The prediction accuracy on the validation set should be at least 50% for this 3-

# possible packages you might need are: scikit-learn, numpy, torch
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import TensorDataset, DataLoader

import torch.optim as optim
```

- Build the train loader and validation loader

```
# convert your numpy array to TensorDataset and create a data loader for training and
# some hyperparameters: input dimension, output dimension, batch size, number of epoc
epochs = 5
lr = 1e-4
indim = X.shape[1]
outdim = 3
drate = 0.7
batch_size = 16
```

- Build the network

```
# create your model/network
class SentimentNetwork(nn.Module):

  def __init__(self, input_dim, output_dim, dropout_rate):

    super(SentimentNetwork,self).__init__()


  def forward(self,x):



    return x

# create a model
model = SentimentNetwork(indim,outdim,drate)
print(model)
```

```
SentimentNetwork(
  (fc1): Linear(in_features=500, out_features=100, bias=True)
  (dropout): Dropout(p=0.7, inplace=False)
  (fc2): Linear(in_features=100, out_features=50, bias=True)
  (fc3): Linear(in_features=50, out_features=3, bias=True)
)
```

- Create a training function to train the model and an evaluation function to evaluate the performance on the separate validation set

```
# define a training process function
def train(model, train_loader, optimizer, criterion):

  epoch_loss, epoch_acc = 0.0,0.0 # the loss and accuracy for each epoch

  model.train()



  return epoch_loss, epoch_acc

# define a validation/evaluation process function
def evaluate(model, val_loader, criterion):

  epoch_loss, epoch_acc = 0.0,0.0 # the loss and accuracy for each epoch

  model.eval()

  with torch.no_grad():
```

```
  with torch.no_grad():


  return epoch_loss, epoch_acc
```

- Main starting point: train the model and evaluate the model

```
# define the loss function and optimizer


# real training and evaluation process
for epoch in range(epochs):
  train_loss, train_acc = train(model, train_loader, optimizer, criterion)
  valid_loss, valid_acc = evaluate(model, val_loader, criterion)

  print(f'Epoch: {epoch+1:02}')
  print(f'\tTrain Loss: {train_loss:.4f} | Train Acc: {train_acc:.4f}')
  print(f'\t Val. Loss: {valid_loss:.4f} |  Val. Acc: {valid_acc:.4f}')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:18: UserWarning: Im
Epoch: 01
        Train Loss: 0.7994 | Train Acc: 0.6475
         Val. Loss: 0.8105 |  Val. Acc: 0.6150
Epoch: 02
        Train Loss: 0.8040 | Train Acc: 0.6475
         Val. Loss: 0.8024 |  Val. Acc: 0.6150
Epoch: 03
        Train Loss: 0.7895 | Train Acc: 0.6475
         Val. Loss: 0.7940 |  Val. Acc: 0.6150
Epoch: 04
        Train Loss: 0.7817 | Train Acc: 0.6475
         Val. Loss: 0.7853 |  Val. Acc: 0.6150
Epoch: 05
        Train Loss: 0.7686 | Train Acc: 0.6469
         Val. Loss: 0.7758 |  Val. Acc: 0.6150
```