Workshop

1 Import the necessary libraries

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn import preprocessing
from sklearn import svm
from sklearn.model_selection import train_test_split
import spacy as sp
import re
import pickle as pkl
from sklearn import metrics
from sklearn.svm import libsvm,SVC
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
```

2 Load the spacy model

[ ]:
```
nlp = sp.load('en_core_web_lg')
```

3 Load the data file and do some preliminary exploration

- Use Panda to read the data csv file
- Check the top 5 raws in the data frame
- Review the unique values of the Category, i.e. the labels
- Get the number of samples per class
- Sample some Snippets

[ ]:
```
df = pd.read_csv('sampled_data.csv',index_col=0)
```

[ ]:
```
df.head()
```

```
[ ]: df.Category.unique()
```

```
[ ]: df.Category.value_counts()
```

```
[ ]: df['Snippet'][10]
```

```
[ ]: df['Snippet'][2000]
```

```
[ ]: df['Snippet'][5000]
```

4 Data Pre-Processing

- Remove the NaNs
- Combine the string data in a new column after converting all the string to lower case
- Clean the text
  - remove non_alphabets
  - use spaCy model to keep Nouns, Verbs and ProNouns
  - use spaCy model to remove non-English words

```
[ ]: df.Title.fillna('',inplace=True)
     df.Snippet.fillna('',inplace=True)
```

```
[ ]: df['Doc']=df.Keyword+' '+ df.Title.str.lower()+' '+ df.Snippet.str.lower()
```

```
[ ]: df['Doc'][100]
```

```
[ ]: def string_preprocessing(data):
         ''' The function processing the data including: keeping only nouns verbs and␣
      ↪pronouns, remove extra characters
         Args:
             data: string
         Returns:
             string
         '''

         data = re.sub('[^a-z]',' ',data)

         doc = nlp(data)
         text = []
         for word in doc:
             if word.pos_ in ['PROPN','NOUN','VERB'] and np.sum(word.vector) !=0:
                 text.append(word.text)

         return ' '.join(text)
```

```
[ ]: df['Doc'][1004]
```

```
[ ]: string_preprocessing(df['Doc'][1004])
```

5 Build LDA

- write functions to build an LDA model and to view the generated topics,
- the model uses CountVectorizer as an input to the LDA model
- build LDA for the class Technology to demonstrate the output

```
[ ]: def build_lda(data,num_topics):
         tf_vectorizer = CountVectorizer(max_df=0.95, min_df=2,
                                         stop_words='english')
         tf = tf_vectorizer.fit_transform(data)

         lda = LatentDirichletAllocation(n_components=num_topics, max_iter=5,
                                         learning_method='online',
                                         learning_offset=50.,
                                         random_state=0)
         lda.fit(tf)

         return lda,tf_vectorizer

     def print_top_words(model, feature_names, n_top_words):
         for topic_idx, topic in enumerate(model.components_):
             message = "Topic #%d: " % topic_idx
             message += " ".join([feature_names[i]
                                 for i in topic.argsort()[:-n_top_words - 1:-1]])
             print(message)
         print()
```

```
[ ]: data_tech = df[df['Category']=='Technology']['Doc'].apply(lambda x:␣
      ↪string_preprocessing(x))
```

```
[ ]: lda_technology,tf_vectorizer = build_lda(data_tech,num_topics=10) # change␣
      ↪num_topics to see its impact
     print_top_words(lda_technology,tf_vectorizer.
      ↪get_feature_names(),n_top_words=10) # change n_top_words to see its impact
```

5.1 Run LDA for every class in the data

- Assume all classes require the same number of topics
- Clean the text before building the LDA models

```
[ ]: n_components = 20

ldas = []
tf_vectorizers = []
for cat in df.Category.unique():
    cat_df = df['Doc'].where(df['Category'] == cat).dropna(how='any')
    cat_df = cat_df.apply(lambda x: string_preprocessing(x))
    lda_t,tf_vectorizer_t = build_lda(cat_df,num_topics=n_components)
    ldas.append(lda_t)
    tf_vectorizers.append(tf_vectorizer_t)
```

6 Save the LDA models

Do not run if you do not want to override the supplied file

```
[ ]: with open('tutorial_ldas.pkl','wb') as f:
        pkl.dump((ldas,tf_vectorizers),f,pkl.HIGHEST_PROTOCOL)
```

6.1 Load the LDA models from an already provided file

```
[ ]: with open('tutorial_ldas.pkl','rb') as f:
       (ldas,tf_vectorizers)= pkl.load(f)
```

7 Feature Engineering

- Pass the data to every LDA models, extract and combine the topic features
- if wordEmbed = True then add the spaCy langauge model embedding to the extract LDA features

```
[ ]: def feature_extraction(data,ldas,tf_vects, wordEmbed=True):
    features=[]
    labels =[]
    assert(len(ldas)==len(tf_vects))
    for i,d in data.iterrows():
        labels.append(d['Category'])
        line= []
        for j in range(len(ldas)):
            line.extend(ldas[j].transform(tf_vects[j].transform([d['Doc']]))[0])
        if wordEmbed:
            line.extend(list(nlp(d['Doc']).vector))
        features.append(line)
    return features,labels
```

```
[ ]: features,labels=feature_extraction(df.dropna(how='any'),ldas,tf_vectorizers)
```

8 Save the extracted features

Do not run if you do not want to override the supplied file

```
[ ]: with open('tutorial_features.pkl','wb') as f:
             pkl.dump((features,labels),f,pkl.HIGHEST_PROTOCOL)
```

8.1 Load the extracted features from the provided file

```
[ ]: with open('tutorial_features.pkl','rb') as f:
            (features,labels)= pkl.load(f)
```

9 Prepare for classification

- Fit a label encoder to convert String labels into numbers

```
[ ]: # encode the labels
     le = preprocessing.LabelEncoder()
     le.fit(labels)
     encoded_labels = le.transform(labels)
     n_classes = len(le.classes_)
```

10 Shallow Classifier

- Split the data into training and testing sets
- Train a linear SVM on the training data
- Provide results of the accuracy on the test data

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(features,␣
      ↪encoded_labels,test_size=0.25)
     clf = SVC(kernel='linear',probability=True)
     clf.fit(X_train,y_train)
```

```
[ ]: print(metrics.classification_report(y_test,clf.predict(X_test)))
```

11 Can you test the shallow classifier on LDA features only?

12 Demo

- Define a function to process a string to be suitable to run against the model
- Use the built SVM model to predict the class
- Output the top three classes with their probabilities

[ ]:
```
def string_features(text,ldas,tf_vectorizers):
        ''' extract lda features for a given text.
        Args:
            text: string
            ldas: a list of LDA models
            tf_vectorizers: a list of CounterVectorizers associated with the ldas
        Returns:
            a list of the lda features with length [number of lda models] X␣
↪[number of topics per model]
        '''
        line= []
        for j in range(len(ldas)):
            line.extend(ldas[j].transform(tf_vectorizers[j].
↪transform([text]))[0])
        vec = nlp(text).vector
        line.extend(list(vec))
        return line
```

[ ]:
```
while True:
    print('Enter a business description please, q to exit:\n')
    st = input()
    if st == 'q':
        break
    clean_st =  string_preprocessing(st)
    feats = string_features(clean_st,ldas,tf_vectorizers)
    probs = clf.predict_proba([feats])[0]
    idx = np.argsort(probs)[::-1]
    top_probs = probs[idx[:3]]
    top_labels = le.inverse_transform(idx[:3])

    for lbl,prob in zip(top_labels,top_probs):
        print(lbl,':',100*prob)
    print ('*****************\n')
```

13 Deep Learning

- Define a Multi-Layer Perceptron to classify the data

- Use Two layer and a softmax layer
- Use Dropout
- Use Relu activation functions

[ ]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout

model = Sequential()
# First Layer
model.add(Dense(1150, input_dim=len(X_train[0])))
model.add(Activation('relu'))
model.add(Dropout(0.5))
#Second Layer
model.add(Dense(500))
model.add(Activation('relu'))
#Third Layer
model.add(Dense(n_classes))
model.add(Activation('softmax'))

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

model.summary()
```

13.1 Convert the labels from a noiminal value to one-hot encoded

this is a requirment to be able to run the keras model

[ ]:
```
from keras.utils import to_categorical
train_label = to_categorical(y_train, num_classes=n_classes)
test_label =  to_categorical(y_test, num_classes=n_classes)
```

13.2 Train the model

- Define a model checkpoint to save the best model through the training iterations
- Define batch size and Epochs

[ ]:
```
from keras.callbacks import ModelCheckpoint
checkpointer = ModelCheckpoint(filepath='tutorial_weights.{epoch:02d}-{val_loss:
 ↪.2f}.hdf5', verbose=1, save_best_only=True,monitor='val_acc')
hist=model.fit(np.asarray(X_train), train_label, epochs=20, batch_size=100,
               validation_data=(np.
 ↪asarray(X_test),test_label),callbacks=[checkpointer])
```

13.3 Plot the output to understand the change of training and validation accuracy over training epochs

```
[ ]: plt.plot(hist.history['acc'])
     plt.plot(hist.history['val_acc'])
```

```
[ ]: preds = np.argmax(model.predict(np.asarray(X_test)),axis=1)
     print(metrics.classification_report(y_test,preds))
```

13.4 Can you test different variations of the model?

13.5 Prepare Data to use in LSTM

- For LSTM the data has to be prepared differently to format it as sequences
- Each element in the sequence is a word embedding from the spaCy language model
- Define max sequence length
- Add zero paddings for shorter seuquences

```
[ ]: from keras.preprocessing.text import Tokenizer
     from keras.preprocessing.sequence import pad_sequences
     from keras.utils import to_categorical
     from keras.layers import Dense, Input, GlobalMaxPooling1D
     from keras.layers import Conv1D, MaxPooling1D, Embedding,LSTM,Bidirectional
     from keras.models import Model,Sequential
     from keras.initializers import Constant
     from keras.callbacks import ModelCheckpoint
```

```
[ ]: MAX_NUM_WORDS = nlp.vocab.length # the number of words in the dictionary
     MAX_SEQUENCE_LENGTH = 10
     EMBEDDING_DIM = 300 # comes from spaCy, if you select a different model this has␣
      ↪to change accordingly
```

```
[ ]: #prepare text samples and their labels
     texts = df['Doc'].apply(lambda x: string_preprocessing(x))
```

13.6 Build a Keras tokenizer and convert text into sequences

```
[ ]: tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
     tokenizer.fit_on_texts(texts)
     sequences = tokenizer.texts_to_sequences(texts)
```

```
[ ]: word_index = tokenizer.word_index
     print('Found %s unique tokens.' % len(word_index))
```

13.7 Add padding if required

[ ]:
```
data_seq = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
```

[ ]:
```
data_seq.shape
```

13.8 Create the embedding layer which is not trainable, this will be the input to the LSTM model

[ ]:
```
# prepare embedding matrix
num_words = min(MAX_NUM_WORDS, len(word_index)) + 1
embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))
for word, i in word_index.items():
    if i > MAX_NUM_WORDS:
        continue
    embedding_vector = nlp(word).vector
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector

# load pre-trained word embeddings into an Embedding layer
# set trainable = False so as to keep the embeddings fixed
embedding_layer = Embedding(num_words,
                            EMBEDDING_DIM,
                            embeddings_initializer=Constant(embedding_matrix),
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False)

print('Training model.')
```

[ ]:
```
#LSTM
batch_size = 100
print('Build model...')
lstm_model = Sequential()
lstm_model.add(embedding_layer)
lstm_model.add(LSTM(200, dropout=0.3, recurrent_dropout=0.
 ↪3,return_sequences=False))
# if you want to add another layer set return_sequences to True
#lstm_model.add(Bidirectional(LSTM(50, dropout=0.5, recurrent_dropout=0.5)))
lstm_model.add(Dense(30, activation='tanh'))
lstm_model.add(Dense(19, activation='softmax'))

# try using different optimizers and different optimizer configs
lstm_model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
print('Train...')
lstm_model.summary()
```

[ ]:
```
# Reformat the labels
seq_labels = to_categorical(encoded_labels,num_classes=n_classes)
```

[ ]:
```
checkpoint = ModelCheckpoint(filepath='weights-improvement-lstm-{epoch:
↪02d}-{val_acc:.2f}.hdf5',
                                          monitor='val_loss', verbose=0,␣
↪save_best_only=True)
hist = lstm_model.fit(data_seq, seq_labels,
          batch_size=batch_size,
          epochs=20,validation_split=0.2,shuffle=True,callbacks=[checkpoint])


score, acc = model.evaluate(data_seq, seq_labels,
                            batch_size=batch_size)
print('Train score:', score)
print('Train accuracy:', acc)
```

[ ]:
```
plt.plot(hist.history['acc'])
plt.plot(hist.history['val_acc'])
plt.show()
```

[ ]:
```
pred = lstm_model.predict_classes(data_seq)
print(classification_report(np.argmax(seq_labels,axis=1),pred))
```

13.9 Is that a better model than an MLP?

- if not what can you change?
- Is the test correct?