Using Scikit learn for Text Classification, A walk through

In [1]: from sklearn.datasets import fetch_20newsgroups
twenty_train = fetch_20newsgroups(subset='train', shuffle=True)

In [2]: twenty_train.target_names #prints all the categories
   print("\n".join(twenty_train.data[0].split("\n")[:3])) #prints first line of the first data file

From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu

Building the vectorizer: count vectorizer indexes all the words in the corpus, and counts them in each document

In [3]: from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
   X_train_counts = count_vect.fit_transform(twenty_train.data)
   X_train_counts.shape

Out[3]: (11314, 130107)

Creating features for each indexed word based on its appearance in different documents, and its total appearance in the whole document set (tfidf features), we use other features in the assignment

In [4]: from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
   X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
   X_train_tfidf.shape

Out[4]: (11314, 130107)

Building a Pipeline (vectorizer, transformer, classifier), the pieline takes in the input document, computes the tfidf features, and then feeds those features to the classifier. In the assignment we are only using the classifier so the interface would be different. You could check the source code for the scikitlearn classifiers for a thorough explanation and a quick and clear understanding. Here we are using the naive bayes classifier. For the assignment you will be using SVC and other ones: you could find the source code for the SVC classifier in here: https://github.com/scikit-learn/scikit-learn/blob/a24c8b46/sklearn/svm/classes.py#L426
In [5]: # from that page: a simple example is:
    import numpy as np
    X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
    y = np.array([1, 1, 2, 2])
from sklearn.svm import SVC
    clf = SVC()
    clf.fit(X, y)
    # or you could specify your classifier parameters:
    # clf = SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    # decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
    # max_iter=-1, probability=False, random_state=None, shrinking=True,
    # tol=0.001, verbose=False)
print(clf.predict([[-0.8, -1]]))

[1]

Now, back to the naive bayes classifier, we build our pipeline and fit the input data, (in this case the training document set) and the output features

In [6]: from sklearn.naive_bayes import MultinomialNB
    clf = MultinomialNB().fit(X_train_tfidf, twenty_train.target)

In [7]: from sklearn.pipeline import Pipeline
    text_clf = Pipeline([('vect', CountVectorizer()),
                        ('tfidf', TfidfTransformer()),
                        ('clf', MultinomialNB())])

In [8]: text_clf = text_clf.fit(twenty_train.data, twenty_train.target)

    and then we can observe the performance

In [9]: import numpy as np
    twenty_test = fetch_20newsgroups(subset='test', shuffle=True)
    predicted = text_clf.predict(twenty_test.data)
    np.mean(predicted == twenty_test.target)

Out[9]: 0.7738980350504514

Now lets try an SVM classifier instead:
Building an SVM classifier

In [10]: from sklearn.linear_model import SGDClassifier

In [11]: text_clf_svm = Pipeline([('vect', CountVectorizer()),
                              ('tfidf', TfidfTransformer()),
                              ('clf-svm', SGDClassifier(loss='hinge', penalty='l2',
                                                        alpha=1e-3, n_iter=5, random_state=42))])
In [ ]: _ = text_clf_svm.fit(twenty_train.data, twenty_train.target)

In [13]: predicted_svm = text_clf_svm.predict(twenty_test.data)

and then computing the accuracy:

In [14]: np.mean(predicted_svm == twenty_test.target)

Out[14]: 0.82381837493361654

Performing the grid search could be useful for tuning hyperparameters, I would try finding a good set of hyperparameters on a small training set/lower number of training iterations, but this does not always result in optimal performance as you use the same hyper params for a larger training set/greater number of iterations

In [15]: from sklearn.model_selection import GridSearchCV
   parameters = {'vect__ngram_range': [(1, 1), (1, 2)],
                  'tfidf__use_idf': (True, False),
                  'clf__alpha': (1e-2, 1e-3),
               }

In [16]: gs_clf = GridSearchCV(text_clf, parameters, n_jobs=-1)
   gs_clf = gs_clf.fit(twenty_train.data, twenty_train.target)

In [17]: print(gs_clf.best_score_)
   print(gs_clf.best_params_)

0.906752695775
{'clf__alpha': 0.01, 'tfidf__use_idf': True, 'vect__ngram_range': (1, 2)}

In [ ]: parameters_svm = {'vect__ngram_range': [(1, 1), (1, 2)],
                          'tfidf__use_idf': (True, False),
                          'clf-svm__alpha': (1e-2, 1e-3),
                       }
   gs_clf_svm = GridSearchCV(text_clf_svm, parameters_svm, n_jobs=-1)
   gs_clf_svm = gs_clf_svm.fit(twenty_train.data, twenty_train.target)

In [19]: print(gs_clf_svm.best_score_)
   print(gs_clf_svm.best_params_)

0.89791408874
{'clf-svm__alpha': 0.001, 'tfidf__use_idf': True, 'vect__ngram_range': (1, 2)}

In [20]: text_clf = Pipeline([(('vect', CountVectorizer(stop_words='english'))),
                       ('tfidf', TfidfTransformer()),
                       ('clf', MultinomialNB()),])
In [ ]: import nltk
    nltk.download()

In [22]: from nltk.stem.snowball import SnowballStemmer
    stemmer = SnowballStemmer("english", ignore_stopwords=True)

    last part: removing the stop words and effect on performance, you will do this one manually in your assignment

In [23]: class StemmedCountVectorizer(CountVectorizer):
    
    def build_analyzer(self):
        
        analyzer = super(StemmedCountVectorizer, self).build_analyzer()
        return lambda doc: ([stemmer.stem(w) for w in analyzer(doc)])

    stemmed_count_vect = StemmedCountVectorizer(stop_words='english')
    text_mnb_stemmed = Pipeline([('vect', stemmed_count_vect),
                                ('tfidf', TfidfTransformer()),
                                ('mnb', MultinomialNB(fit_prior=False))],
                                )
    text_mnb_stemmed = text_mnb_stemmed.fit(twenty_train.data, twenty_train.target)
    predicted_mnb_stemmed = text_mnb_stemmed.predict(twenty_test.data)
    np.mean(predicted_mnb_stemmed == twenty_test.target)

Out[23]: 0.81678173127987252

This document was based on this medium post: https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a

It might be a good idea to mount the remote folder on the sever on your computer, in order not to have to deal with multiple version at the same time. You can then edit your files in pycharm on your local machine or any other ide that you would prefer. For ubuntu it is quite straightforward to setup. For mac users I would recommend using osx fuse and here is an example how to use it:

    sshfs [-p] [port number] remote_user@remote_host:remote_path local_host_path

For windows I have not found a good tool to the same, although filezilla and putty give a nice combination

A few suggestions regarding the speed of your program (also posted on Piazza):

A few things regarding the runtime of the code: I would suggest checking the functionality of your code on a small set and see if it works as you expect. If your code is running slow you could use python profiler to find out which part is actually slowing it down, sometimes it is the way you handle the data structures (like list vs dictionary, etc) and then you can speed it up if you want. As professor kindly suggested, you could use tmux to leave the process running for as long as you want, so as long as you know it works correctly, there is no problem in running it for longer periods. It is fairly simple to convert your python code to cython, and if done right, that could also help a little bit with the speed up. Multiprocessing/Multithreading is quite useful in case process can be run in parallel (especially useful for preprocessing steps)

Hope it helps, any one who has other suggestions please share your opinion.