Solution to question 5

In [98]:
```
import numpy as np
import sklearn
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans

twenty_train = fetch_20newsgroups(subset='train', shuffle=True)
twenty_test = fetch_20newsgroups(subset='test', shuffle=True)

print(twenty_train.keys())

```

```
dict_keys(['data', 'filenames', 'target_names', 'target', 'DESCR'])
```

Using the guide given in https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a (https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a) The data is in key 'data' with the target labels in key 'target'

As sklearn already provides the data in the form of train and test data, using 300 samples from the training data for now

In [99]:
```
data = twenty_train.data
data = data[0:300]
```

Removed the stop words while using TfIdfVectorizer

In [100]:
```
vectorizer = TfidfVectorizer(stop_words = 'english')
X = vectorizer.fit_transform(data)
# print(vectorizer.get_feature_names())
```

In [101]:
```
true_k = 5
model = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n
model.fit(X)
```

Out[101]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=100,
       n_clusters=5, n_init=1, n_jobs=None, precompute_distances='aut
       random_state=None, tol=0.0001, verbose=0)
```

In [102]:
```python
print("Top terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()
for i in range(true_k):
    print("\nCluster %d:" % i),
    for ind in order_centroids[i, :1]:
        print(' %s' % terms[ind]),
    print
```

```
Top terms per cluster:

Cluster 0:
 edu

Cluster 1:
 edu

Cluster 2:
 edu

Cluster 3:
 edu

Cluster 4:
 com
```

In [103]:
```python
print("\n")
print("Prediction")

Y = vectorizer.transform(["Android surpasses iOS."])
prediction = model.predict(Y)
print("the class of the above news is ", twenty_train.target_names[

Y = vectorizer.transform(["Joe Biden defeats Trump"])
prediction = model.predict(Y)
print("the class of the above news is ", twenty_train.target_names[
```

```


Prediction
the class of the above news is  comp.graphics
the class of the above news is  comp.graphics
```

I am not sure if it is working correctly as it classifies the political news as ms-windows.misc

Solution to question 6, based on examples provided in http://www.science.smith.edu/~jcrouser/SDS293/labs/lab7-py.html (http://www.science.smith.edu/~jcrouser/SDS293/labs/lab7-py.html)

In [104]:
```python
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
data = pd.read_excel("ENB2012_data.xlsx")
```

In [105]:
```python
data.head()
```

Out[105]:

	X1	X2	X3	X4	X5	X6	X7	Y
0	0.74	686.0	245.0	3.5	2	0.0	0	0
1	0.74	686.0	245.0	3.5	4	0.0	0	0
2	0.74	686.0	245.0	3.5	5	0.0	0	0
3	0.74	686.0	245.0	3.5	3	0.0	0	0
4	0.71	710.5	269.5	3.5	2	0.0	0	0

In [106]:
```python
X = data.iloc[:,1:6]
y = data.iloc[:,7]
```

In [107]:
```python
x_train, x_test, y_train, y_test = train_test_split(X, y)
```

In [108]:
```python
clf = LinearRegression()
scores = cross_val_score(clf, x_train, y_train, cv=5)
# scores = cross_val_score(clf, X, y, cv=5)
```

In [109]:
```python
print("average cross validation accuracy is ",scores.mean())
```

```
average cross validation accuracy is  0.9139580130418947
```

After cross validation, the classifer performs 90% correctly in the training data. Now the classifer will be tested on the testing data. The data will be fitted to the x_train and y_train and confusion matrix will be shown w.r.t the tessting data.

In [110]:
```python
clf.fit(x_train, y_train)
y_pred_linear = clf.predict(x_test)
```

In [ ]:
```python

```

Need to convert the output of regression into binary outputs for calculating accuracy

In [111]:
```python
y_pred_linear_new = [0 if x <0.5 else 1 for x in y_pred_linear]
```

In [112]:
```python
print(confusion_matrix(y_test, y_pred_linear_new))
```

```
[[97  0]
 [ 6 89]]
```

Now we do the same process for logistic regression

In [113]:
```
clf = LogisticRegression()
scores = cross_val_score(clf, x_train, y_train, cv=5)
# scores = cross_val_score(clf, X, y, cv=5)
print("average cross validation accuracy is ",scores.mean())
clf.fit(x_train, y_train)
y_pred_logistic = clf.predict(x_test)
y_pred_logistic_new = [0 if x <0.5 else 1 for x in y_pred_logistic]
print(confusion_matrix(y_test, y_pred_logistic_new))
```

```
average cross validation accuracy is  0.9773442226255293
[[97  0]
 [ 6 89]]
```

After doing the corss validation, average accuracy for logistic regression is at 97% which is more than linear regression's 90%.

Confusion matrix can be calculated for each model so we can calculate it for each fold in the corss validation. However here the confusion matrix was calculated directly using the testing data without involving the cross validation. Due to this, there is a difference in the cross validation accuracy reported on the training data (90% vs 97%) while the confusion matrix over the testing data remains the same. The testing data was not involved in the CV process. The confusion matrix has 192 elements which is 25% of the total 768 elements as we used standard 80-20 split for the train and testing data.