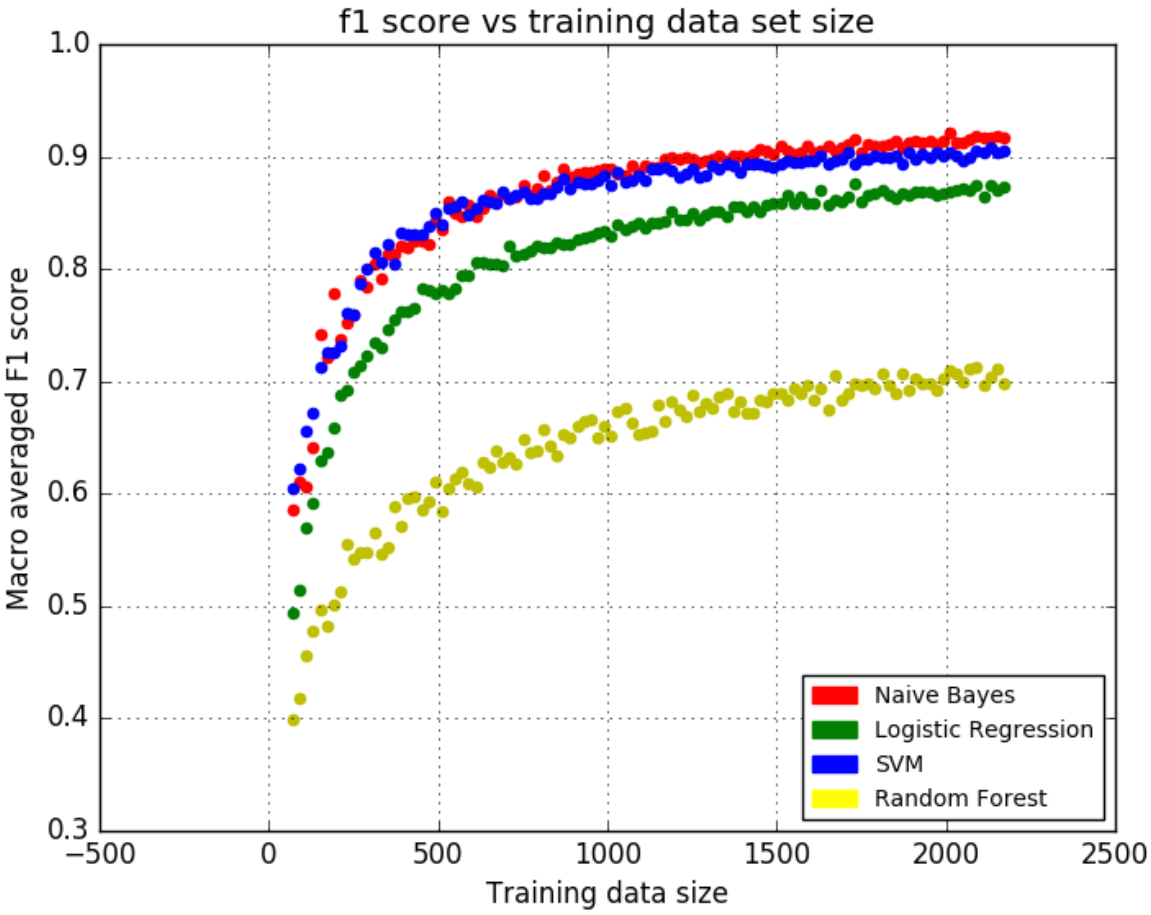
1a) File to run : part1a.py

Classifier	Feature extraction	ngram	precision	recall	f1-score
NaiveBayes	count vector	1	0.93673841024	0.911195398933	0.918924253602
NaiveBayes	tfidf	1	0.890397780998	0.755247006175	0.736478056409
NaiveBayes	count vector	2	0.905230614104	0.870762691686	0.879082682678
NaiveBayes	tfidf	2	0.883432802981	0.78731224233	0.790665221004
LogisticRegression	count vector	1	0.901394625581	0.892575588199	0.895792284318
LogisticRegression	tfidf	1	0.876021386517	0.838605340134	0.842783633464
LogisticRegression	count vector	2	0.827631026405	0.815561906905	0.818032888502
LogisticRegression	tfidf	2	0.69670580148	0.639085322371	0.634241820239
SVM	count vector	1	0.895878620943	0.883819119288	0.887812299766
SVM	tfidf	1	0.933327500547	0.909739707778	0.916760374895
SVM	count vector	2	0.889573541907	0.859929980392	0.867411643842
SVM	tfidf	2	0.922431030911	0.8925182951	0.900485075422
RandomForest	count vector	1	0.79694418812	0.738689002939	0.731141057159
RandomForest	tfidf	1	0.811323114482	0.755531388285	0.751213609588
RandomForest	count vector	2	0.800246192688	0.726565783192	0.7318848658 4
RandomForest	tfidf	2	0.77708295114	0.701761294032	0.700993804776

Best feature extractor with unigram:to be used for part 1b
Naive Bayes:count vector
Logistic Regression: count vector
SVM:tfidf
Random Forest:tfidf

1b) File to run: part1b.py. The data that was used to plot tis graph is there in 1gp_plot.txt, 2gp_plot.txt, 3gp_plot.txt, 4gp_plot.txt.



1c)
Describe your findings and make arguments to explain why this is the

case. You can use any online source to understand the relative performance of algorithms for varying training data sizes. Make arguments in your own words. Provide a citation.

As seen in the graph for part1b, Naive Bayes performs marginally better than SVM, so I used the following 3 configurations to resolve what looks like a tie:
1) config1: lower case and filter out stop words
2) config2: lower case and Porter Stemmer
3) config3: both config1 and config2

Since these are the only optimizations(out of the ones listed for the assignment) that can be applied to Naive Bayes, its highest f1 score should be greater than that of SVM in order to conclude with certainty that Naive Bayes is better than SVM for this particular classification task. However, for Naive Bayes I get the following f1 scores:
0.928971241563 - config1
0.923321085209 - config2
0.90983049155 - config3

while for SVM I get:
0.932705821822 - config1
0.913638468779 - config2
0.894790186183 - config3

The highest f1 score of SVM beats the highest f1 score for Naive Bayes.

This shows that SVM with tweaking, can perform significantly better than Naive Bayes which is why I will go with SVM as the better classifier for part2.

Poor performance of Random Forest: Random Forest functions by choosing a sample out of all data points, and then constructing decision tree by selecting a subset of features at each node and splitting only on those features. If we randomly select a small subset of features, the features that are informative or topic-related might be missed, due to which random forest probably does not do so well in text classification. The discriminative strength of the trees is low.
(source:https://pdfs.semanticscholar.org/9b2f/84d85e5b6979bf375a2d4b15f7526597fc70.pdf)

2a)
File to run the different configurations: part2a.py
The configurations I experimented with are as follows:
Conclusion: Best performing classifier is LinearSVC with l2 regularizer and removal of stop words prior to training.

2b)
The file to run should be of format as below:

```
from sklearn.externals import joblib


categories = ['rec.sport.hockey','sci.med','soc.religion.christian','talk.religion.misc']

twenty_test = load_files(container_path='Test',categories=categories,load_content=True,encoding='latin-1')



clf,count_vect,tfidf_transformer = joblib.load('classifier.pkl')

X_new_counts = count_vect.transform(twenty_test.data)
X_new_tfidf = tfidf_transformer.transform(X_new_counts)

predicted = clf.predict(X_new_tfidf)
```

In addition to the required libraries, also import joblib from sklearn.externals. Then deserialize classifier(clf), count vectorizer(count_vect) and tfidf transformer as shown using joblib.load and use it as required.

If you want to use the training data to train a new classifier, use training_data.pkl instead to get the data. For example:
Trained_data = joblib.load('trained_data.pkl') and then use it as required.

2c)
SVMs are based on the theory of Structural Risk Minimization which means it tries to find a classifier with the lowest probability of erroneously predicting a new sample. It measures the complexity of the classifier based on how wide a margin can separate the data and not the number of features. That makes them independent of the dimension of feature space which works out well in case of text data that has a high number of features. Other classifiers usually try to avoid high feature data space by reducing the number of features being considered. However, in text data there are very few irrelevant features, and often a classifier that trains only on features with low count value, performs better than the ones that don't consider them at all. Lastly, text data is linearly separable which is the sole target of SVMs- to find linear separability.
(source: http://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf)