|  |  |
| --- | --- |
|  |  |
|  | # In[4]: |
|  |  |
|  | # You can check the target names (categories) and some data files by following commands. |
|  | twenty\_train.target\_names #prints all the categories |
|  |  |
|  |  |
|  | # In[5]: |
|  |  |
|  | print("\n".join(twenty\_train.data[0].split("\n")[:3])) #prints first line of the first data file |
|  |  |
|  |  |
|  | # In[6]: |
|  |  |
|  | # Extracting features from text files |
|  | from sklearn.feature\_extraction.text import CountVectorizer |
|  | count\_vect = CountVectorizer() |
|  | X\_train\_counts = count\_vect.fit\_transform(twenty\_train.data) |
|  | X\_train\_counts.shape |
|  |  |
|  |  |
|  | # In[7]: |
|  |  |
|  | # TF-IDF |
|  | from sklearn.feature\_extraction.text import TfidfTransformer |
|  | tfidf\_transformer = TfidfTransformer() |
|  | X\_train\_tfidf = tfidf\_transformer.fit\_transform(X\_train\_counts) |
|  | X\_train\_tfidf.shape |
|  |  |
|  |  |
|  | # In[9]: |
|  |  |
|  | # Machine Learning |
|  | # Training Naive Bayes (NB) classifier on training data. |
|  | from sklearn.naive\_bayes import MultinomialNB |
|  | clf = MultinomialNB().fit(X\_train\_tfidf, twenty\_train.target) |
|  |  |
|  |  |
|  | # In[14]: |
|  |  |
|  | # Building a pipeline: We can write less code and do all of the above, by building a pipeline as follows: |
|  | # The names ‘vect’ , ‘tfidf’ and ‘clf’ are arbitrary but will be used later. |
|  | # We will be using the 'text\_clf' going forward. |
|  | from sklearn.pipeline import Pipeline |
|  |  |
|  | text\_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), ('clf', MultinomialNB())]) |
|  |  |
|  | text\_clf = text\_clf.fit(twenty\_train.data, twenty\_train.target) |
|  |  |
|  |  |
|  | # In[15]: |
|  |  |
|  | # Performance of NB Classifier |
|  | 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) |
|  |  |
|  |  |
|  | # In[16]: |
|  |  |
|  | # Training Support Vector Machines - SVM and calculating its performance |
|  |  |
|  | from sklearn.linear\_model import SGDClassifier |
|  | text\_clf\_svm = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), |
|  | ('clf-svm', SGDClassifier(loss='hinge', penalty='l2',alpha=1e-3, n\_iter=5, random\_state=42))]) |
|  |  |
|  | text\_clf\_svm = text\_clf\_svm.fit(twenty\_train.data, twenty\_train.target) |
|  | predicted\_svm = text\_clf\_svm.predict(twenty\_test.data) |
|  | np.mean(predicted\_svm == twenty\_test.target) |
|  |  |
|  |  |
|  | # In[18]: |
|  |  |
|  | # Grid Search |
|  | # Here, we are creating a list of parameters for which we would like to do performance tuning. |
|  | # All the parameters name start with the classifier name (remember the arbitrary name we gave). |
|  | # E.g. vect\_\_ngram\_range; here we are telling to use unigram and bigrams and choose the one which is optimal. |
|  |  |
|  | 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[19]: |
|  |  |
|  | # Next, we create an instance of the grid search by passing the classifier, parameters |
|  | # and n\_jobs=-1 which tells to use multiple cores from user machine. |
|  |  |
|  | gs\_clf = GridSearchCV(text\_clf, parameters, n\_jobs=-1) |
|  | gs\_clf = gs\_clf.fit(twenty\_train.data, twenty\_train.target) |
|  |  |
|  |  |
|  | # In[23]: |
|  |  |
|  | # To see the best mean score and the params, run the following code |
|  |  |
|  | gs\_clf.best\_score\_ |
|  | gs\_clf.best\_params\_ |
|  |  |
|  | # Output for above should be: The accuracy has now increased to ~90.6% for the NB classifier (not so naive anymore! 😄) |
|  | # and the corresponding parameters are {‘clf\_\_alpha’: 0.01, ‘tfidf\_\_use\_idf’: True, ‘vect\_\_ngram\_range’: (1, 2)}. |
|  |  |
|  |  |
|  | # In[24]: |
|  |  |
|  | # Similarly doing grid search for SVM |
|  | from sklearn.model\_selection import GridSearchCV |
|  | 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) |
|  |  |
|  |  |
|  | gs\_clf\_svm.best\_score\_ |
|  | gs\_clf\_svm.best\_params\_ |
|  |  |
|  |  |
|  | # In[25]: |
|  |  |
|  | # NLTK |
|  | # Removing stop words |
|  | from sklearn.pipeline import Pipeline |
|  | text\_clf = Pipeline([('vect', CountVectorizer(stop\_words='english')), ('tfidf', TfidfTransformer()), |
|  | ('clf', MultinomialNB())]) |
|  |  |
|  |  |
|  | # In[26]: |
|  |  |
|  | # Stemming Code |
|  |  |
|  | import nltk |
|  | nltk.download() |
|  |  |
|  | from nltk.stem.snowball import SnowballStemmer |
|  | stemmer = SnowballStemmer("english", ignore\_stopwords=True) |
|  |  |
|  | 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) |
|  |  |
|  |  |
|  | # In[ ]: |
|  |  |