MULTI CLASS TEXT CLASSIFICATION MODELS COMPARISON

PROJECT REPORT

Submitted by

KHIVENDER SHEKHAWAT 16BCE0439

CHARITH SARVAREDDY 16BCE0902

Guided by

PROF. SHARMILA BANU

Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Vellore-632014, Tamil Nadu, India

School of Computer Science and Engineering

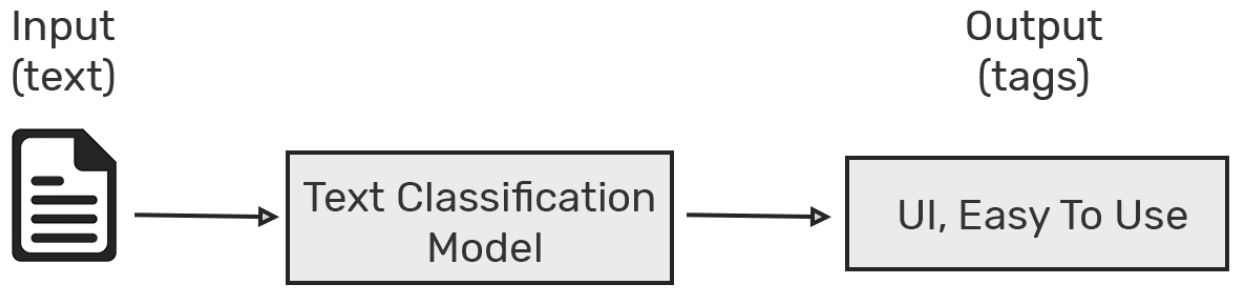
INTRODUCTION

Text classification is the process of conveying tags or categories to text conferring to its content. It's unique of the fundamental tasks in Natural Language Processing (NLP) with extensive applications such as sentiment analysis, topic labeling, spam detection, and intent detection.

Unstructured data in the arrangement of text is all over: emails, chats, web pages, social media, support tickets, survey responses, and more.

Text classification (a.k.a. text categorization or text tagging) is the job of assigning a usual of predefined categories to permitted-text. Text classifiers can be used to shape, structure, and categorize attractive much anything. For example, new articles can be prepared by topics, sustenance tickets can be organized by urgency, chat discussions can be organized by dialectal

A classifier can take this text as an input, analyze its content, and then and automatically assign relevant tags, such as UI and Easy To Use that represent this text:



How Does Text Classification Work?

Text classification can be done in two unlike ways: manual and unconscious classification. In the former, a human annotator understands the content of text and categorizes it therefore. This process usually can provide quality outcomes but it's time-consuming and expensive. The latter applies machine learning, natural language processing, and other systems to automatically classify transcript in a faster and more cost-actual way.

There are many approaches to automatic text classification, which can be grouped into three different types of systems:

- Rule-based systems
- Machine Learning based systems
- Hybrid systems

Rule-based Systems

Rule-based approaches classify text into prepared groups by using a set of handcrafted dialectal rules. These rules educate the system to use semantically applicable elements of a text to recognize relevant categories built on its content. Each rule comprises of an antecedent or decoration and a predicted category.

Say that you want to classify news articles into 2 groups, namely, Sports and Politics. First, you'll need to define two lists of words that characterize each group (e.g. words related to sporting such as football, basketball, LeBron James, etc., and words associated to politics such as Donald Trump, Hillary Clinton, Putin, etc.). Next, when you want to classify a new external text, you'll need to amount the number of sport-related disputes that appear in text and do the same for politics-related words. If the number of sport-related expression appearances is superior than the number of policy-related word count, then the text is classified as sports and vice versa.

For example, this rule-based system will classify the headline "When is LeBron James' first game with the Lakers?" as Sports because it counted 1 sport-related term (Lebron James) and it didn't count any politics-related terms.

Rule-based systems are human logical and can be improved over time. But this method has some disadvantages. For antipasti, these systems necessitate deep knowledge of the domain. They are also time-consuming, since producing rules for a compound system can be quite interesting and usually requires a lot of investigation and testing. Rule-based systems are also tough to maintain and don't scale well given that calculation new rules can disturb the results of the pre-existing rules.

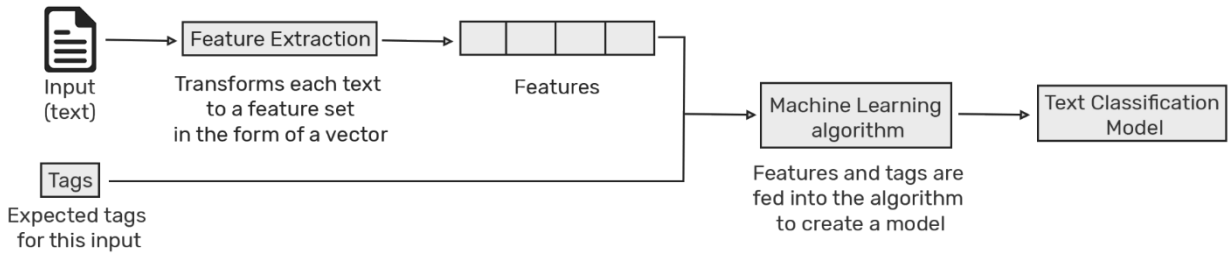
Machine Learning Based Systems

Instead of trusting on manually fashioned instructions, text classification with machine learning absorbs to make classifications based on past observations. By using pre-labeled samples as training data, a machine erudition algorithm can learn the unalike associations between pieces of text and that a precise output (i.e. tags) is expected for a precise input (i.e. text).

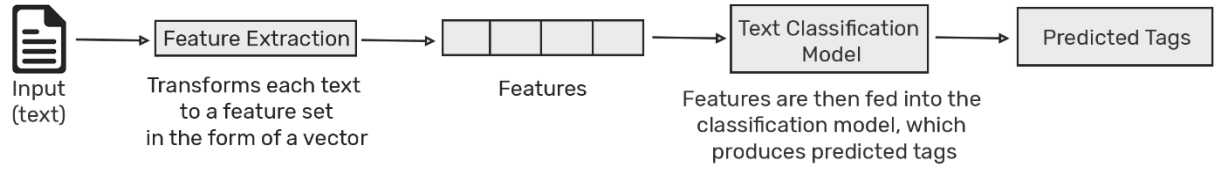
The first step towards drill a classifier with machine learning is feature extraction: a technique is used to transform respectively text into a numerical depiction in the form of a vector. One of the most commonly used approaches is bag of words, where a vector characterizes the frequency of a expression in a predefined dictionary of words.

For example, if we have defined our glossary to have the succeeding words {This, is, the, not, breathtaking, bad, basketball}, and we sought to vectorize the text "This is awesome", we would have the subsequent vector representation of that text: (1, 1, 0, 0, 1, 0, 0).

Then, the machine learning algorithm is nursed with training data that comprises of pairs of feature sets (vectors for each text example) and tickets (e.g. sports, politics) to produce a cataloguing model:



Once it's trained with adequate training samples, the machine education model can begin to make exact predictions. The same feature extractor is used to transmute unseen text to feature sets which can be fed into the classification typical to get calculations on tags (e.g. sports, politics):



Text classification with machine education is usually much additional accurate than human-crafted rule arrangements, especially on complex arrangement tasks. Also, classifiers with machine learning are calmer to maintain and you can always tag new specimens to learn new tasks.

Text Classification Algorithms

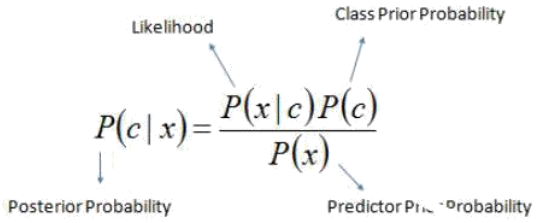
Some of the most prevalent machine learning algorithms for producing text classification representations include the naive bayes personal of algorithms, support vector machines, and deep learning.

Naive Bayes

Naive Bayes is a family of arithmetic algorithms we can make use of when liability text classification. One of the associates of that intimate is Multinomial Naive Bayes (MNB). One of its main recompenses is that you can get really good consequences when data accessible is not much (~ a couple of thousand tagged samples) and computational properties are scarce.

All you need to know is that Naive Bayes is grounded on Bayes's Theorem, which benefits us compute the conditional likelihoods of occurrence of two events grounded on the probabilities of occurrence of each distinct event. This means that any vector that characterizes a text will have to contain material about the probabilities of attendance of the words of the text indoors the texts of a given grouping so that the algorithm can compute the possibility of that text's fitting to the category.

$P(c|x)$, from $P(c)$, $P(x)$, and $P(x|c)$. Naive Bayes classifier assume that the effect of the value of a predictor (x) on a given class (c) is independent of the values of other predictors. Th is assumption is called class conditional independence.



$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

P(c|x) is the posterior probability of class (target) given predictor (attribute).
P(c) is the prior probability of class.
P(x|c) is the likelihood which is the probability of predictor given class.
P(x) is the prior probability of predictor.

The class with the highest posterior probability is the outcome of prediction.

Support Vector Machines

Support Vector Machines (SVM) is just one out of countless algorithms we can indicate from when doing text classification. Like naive bayes, SVM doesn't need much exercise data to twitch providing correct results. Although it needs more computational properties than Naive Bayes, SVM can accomplish more accurate results.

The cost is 0 if the predicted value and the actual value are of the same sign. If they are not, we then calculate the loss value. We also add a regularization parameter the cost function. The objective of the regularization parameter is to balance the margin maximization and loss

$$\frac{\delta}{\delta w_k}\lambda \parallel w \parallel^2 \quad = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k}\left(1 - y_i\langle x_i, w\rangle\right)_+ = \begin{cases} 0, & \text{if } y_i\langle x_i, w\rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

When there is no misclassification, i.e. the model correctly predicts the class of our data point, only update the gradient from the regularization parameter

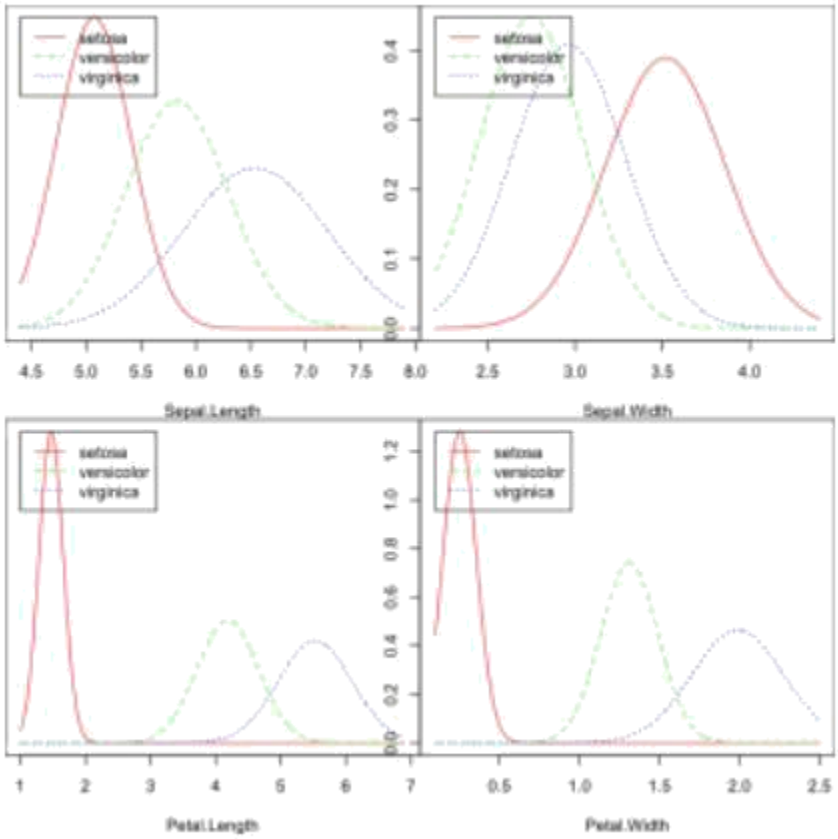
$$w = w - \alpha \cdot (2\lambda w)$$

When there is a misclassification, i.e. the model makes a mistake on the prediction of the class of our data point, the loss along with the regularization parameter to perform gradient update.

Logistic Regression

It is one of the widely used classification algorithm. This algorithm is used in medical as well as business fields for analytics and classification. This model has a hypothesis function $0 \leq h(x) \leq 1$.Where $h\theta(x) = 11 + e\text{-}\theta Tx$ called as Sigmoid or Logistic Function.

The output of this classifier will be a probability of the given input belonging to class 1 If $h\theta(x)$ outputs 0.7 it means that the given input has 70% chance of belonging to class 1. Since we have 10 genre classes $y \in \{0, 1 .. 9\}$ we used one-vs-all method for classification



Metrics and Evaluation

Cross-validation is a joint method to evaluate the presentation of a text classifier. It contains in splitting the training dataset arbitrarily into equal-length circles of examples (e.g. 4 sets with 25% of the statistics). For each set, a text classifier is

qualified with the remaining examples (e.g. 75% of the models). Next, the classifiers make predictions on their individual sets and the results are associated against the human-annotated labels. This allows finding when a calculation was accurate (true positives and true negatives) and when it made a mistake (false positives, false negatives).

With these results, you can figure performance metrics that are valuable for a quick assessment on how well a classifier all of it:

- **Accuracy:** the percentage of texts that were forecast with the correct label.
- **Precision:** the percentage of examples the classifier got right out of the total amount of examples that it predicted for a assumed tag.
- **Recall:** the percentage of instances the classifier predicted for a specified tag out of the total number of examples it should have projected for that given tag.
- **F1 Score:** the harmonic means of precision and recall.

Why is Text Classification Important?

According to IBM, it is projected that around 80% of all information is unstructured, with manuscript being one of the most mutual types of unstructured data. Because of the disordered nature of text, analyzing, understanding, organizing, and sorting through manuscript data is hard and laborious so most companies flop to extract value from that.

This is where text classification with machine learning stages in. By using text classifiers, companies can building business information like as email, legal documents, web pages, chat conversations, and social media messages in a reckless and lucrative way. This allows companies to save time when examining text data, help notify business decisions, and mechanize business processes.

Some of the explanations why companies are leveraging text classification with machine learning are the following:

Scalability

Physically analyzing and organizing text takes time. It’s a sluggish process where a human needs to read every text and decide how to building it. Machine learning vagaries this and enables to easily examine millions of texts at a portion of a cost.

Real-time analysis

There are dangerous situations that companies need to recognize as soon as possible besides take instantaneous action (e.g. PR crises on social media). Text classifiers with appliance learning can make truthful precisions in real-time that empower companies to identify dangerous information instantly and take exploit accurate away.

Consistent criteria

Human annotators make blunders when classifying text data due to disruptions, fatigue, and boredom. Other blunders are generated due to unreliable criteria. In contrast, machine learning applies the same lens and principles to all of the data, thus agreeing humans to reduce errors with national text classification models.

Text Classification Applications

Sentiment Analysis

Probably the most communal example of text classification is sentiment analysis: the electronic process of determining whether a manuscript is positive, negative, or neutral. Companies are using sentiment classifiers on a extensive range of applications, such as invention analytics, brand nursing, customer support, market research, personnel analytics, and much more

Topic Labeling

Another common instance of text classification is topic labeling, that is, sympathetic what a given transcript is talking about. It’s often used for arranging and organizing information such as organizing purchaser feedback by its topic or establishing news articles according to their subject.

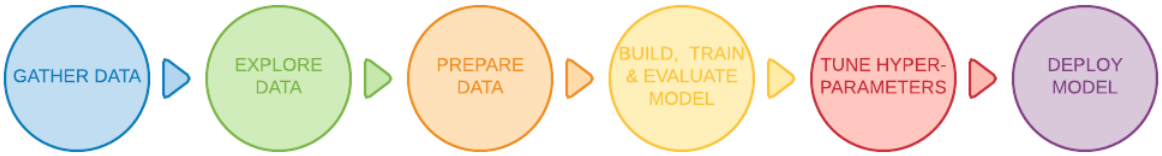
Language Detection

Language detection is additional great example of text classification, that is, the development of classifying received text according to its language. These typescript classifiers are frequently used for routing determinations (e.g. route support vouchers according to their linguistic to the appropriate team).

Intent Detection

Companies are also consuming text classifiers for automatically sensing the intent from customer discussions, often used aimed at generating product analytics or mechanizing business purposes.

Text Classification Workflow



We try dissimilar algorithms and techniques to pursuit for models to produce universal hypotheses, which then make the greatest accurate guesses thinkable about future instances. The same principles smear to text (or document) classification where there are several models can be used to train a text classifier.

The Data

We are using a relatively large data set of Stack Overflow questions and tags. The data is available in Google BigQuery, it is also publicly available at this Cloud Storage URL: https://storage.googleapis.com/tensorflow-workshop-examples/stack-overflow-data.csv.

Exploring the Data

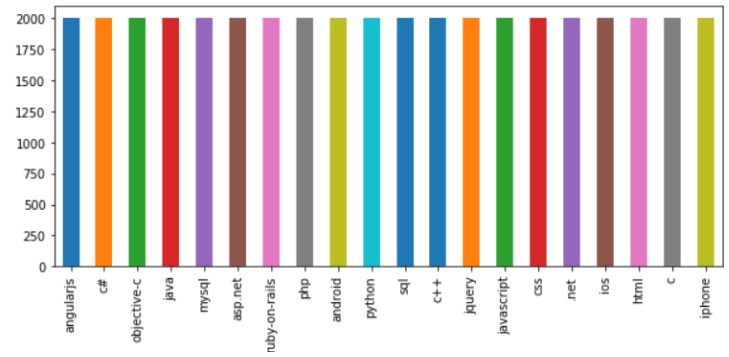
In [1]:
```
import logging
import pandas as pd
import numpy as np
from numpy import random
import nltk
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
import re
from bs4 import BeautifulSoup

%matplotlib inline
```

Out[2]:

	post	tags
0	what is causing this behavior in our c# datet...	c#
1	have dynamic html load as if it was in an ifra...	asp.net
2	how to convert a float value in to min:sec i ...	objective-c
3	.net framework 4 redistributable just wonderi...	.net
4	trying to calculate and print the mean and its...	python

Out[3]: 10286120

In [7]:
```
my_tags = ['java','html','asp.net','c#','ruby-on-rails','jquery','mysql','php','ios','javascript','python','c','css','android',
plt.figure(figsize=(10,4))
df.tags.value_counts().plot(kind='bar');
```



We want to have a look a few post and tag pairs

In [8]:
```
def print_plot(index):
    example = df[df.index == index][['post', 'tags']].values[0]
    if len(example) > 0:
        print(example[0])
        print('Tag:', example[1])
```

In [9]:
```
print_plot(10)
```

```
when we need interface c# <blockquote>    <strong>possible duplicate:</strong><br>   <a href= https://stackoverflow.com/questio
ns/240152/why-would-i-want-to-use-interfaces >why would i want to use interfaces </a>   <a href= https://stackoverflow.com/ques
tions/9451868/why-i-need-interface >why i need interface </a>    </blockquote>     i want to know where and when to use it
for example    <pre><code>interface idemo {  // function prototype  public void show(); }  // first class using the interface c
lass myclass1 : idemo {  public void show()  {   // function body comes here   response.write( i m in myclass );  }  }  // seco
nd class using the interface class myclass2 : idemo {  public void show()   {   // function body comes here   response.write( i
m in myclass2 );   response.write( so  what  );  } </code></pre>   these two classes has the same function name with different
body. this can be even achieved without interface. then why we need an interface where and when to use it
Tag: c#
```

In [10]:
```
print_plot(30)
```

```
how to chain expressions inside ngclass when using the {...}[] form  how can i add another expression to an <code>ng-class</cod
e> directive that uses this form:   <pre><code>ng-class= {true: loading   false: loading-done }[data.loader===null]  </code></p
re>   i d like to add something like this to the list:   <pre><code>{highlight:isspecial} </code></pre>   is it possible withou
t expanding the first expression     thanks.
Tag: angularjs
```

Text Pre-processing

For this particular data set, our text cleaning step includes HTML decoding, remove stop words, change text to lower case, remove punctuation, remove bad characters, and so on.

In [11]:
```
REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;]')
BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')
STOPWORDS = set(stopwords.words('english'))

def clean_text(text):
    """
        text: a string

        return: modified initial string
    """
    text = BeautifulSoup(text, "lxml").text
    text = text.lower()
    text = REPLACE_BY_SPACE_RE.sub(' ', text)
    text = BAD_SYMBOLS_RE.sub('', text)
    text = ' '.join(word for word in text.split() if word not in STOPWORDS)
    return text
```

Now we can have a look a cleaned post

need interface c# possible duplicate would want use interfaces need interface want know use example interface idemo function pr ototype public void show first class using interface class myclass1 idemo public void show function body comes responsewrite my class second class using interface class myclass2 idemo public void show function body comes responsewrite myclass2 responsewri te two classes function name different body even achieved without interface need interface use
Tag: c#

In [14]:
```
df['post'].apply(lambda x: len(x.split(' '))).sum()
```

Out[14]: 3424297

Now we have over 3 million words to work with.

After splitting the data set, the next steps includes feature engineering. We will convert our text documents to a matrix of token counts (CountVectorizer), then transform a count matrix to a normalized tf-idf representation (tf-idf transformer).

```
In [15]: X = df.post
         y = df.tags
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state = 42)
```

Naive Bayes Classifier for Multinomial Models

```
In [13]: from sklearn.naive_bayes import MultinomialNB
         from sklearn.pipeline import Pipeline
         from sklearn.feature_extraction.text import TfidfTransformer

         nb = Pipeline([('vect', CountVectorizer()),
                        ('tfidf', TfidfTransformer()),
                        ('clf', MultinomialNB()),
                       ])
         nb.fit(X_train, y_train)
```

```
Out[13]: Pipeline(memory=None,
              steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                 dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                 lowercase=True, max_df=1.0, max_features=None, min_df=1,
                 ngram_range=(1, 1), preprocessor=None, stop_words=None,
                 strip...inear_tf=False, use_idf=True)), ('clf', MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True))])
```

```
In [14]: %%time
         from sklearn.metrics import classification_report
         y_pred = nb.predict(X_test)

         print('accuracy %s' % accuracy_score(y_pred, y_test))
         print(classification_report(y_test, y_pred,target_names=my_tags))
```

```
accuracy 0.7395
                precision    recall  f1-score   support

          java       0.63      0.65      0.64       613
          html       0.94      0.86      0.90       620
       asp.net       0.87      0.92      0.90       587
            c#       0.70      0.77      0.73       586
   ruby-on-rails       0.73      0.87      0.79       599
        jquery       0.72      0.51      0.60       589
         mysql       0.77      0.74      0.75       594
           php       0.69      0.89      0.78       610
           ios       0.63      0.59      0.61       617
    javascript       0.57      0.65      0.60       587
        python       0.70      0.50      0.59       611
             c       0.79      0.78      0.79       594
           css       0.84      0.59      0.69       619
       android       0.66      0.84      0.74       574
        iphone       0.64      0.83      0.72       584
           sql       0.66      0.64      0.65       578
  objective-c       0.79      0.77      0.78       591
           c++       0.89      0.83      0.86       608
     angularjs       0.94      0.89      0.91       638
          .net       0.74      0.66      0.70       601

     micro avg       0.74      0.74      0.74     12000
     macro avg       0.74      0.74      0.74     12000
  weighted avg       0.75      0.74      0.74     12000

Wall time: 1.2 s
```

We achieved 74% accuracy.

Linear Support Vector Machine

Linear Support Vector Machine is widely regarded as one of the best text classification algorithms.

In [15]:
```
from sklearn.linear_model import SGDClassifier

sgd = Pipeline([('vect', CountVectorizer()),
                ('tfidf', TfidfTransformer()),
                ('clf', SGDClassifier(loss='hinge', penalty='l2',alpha=1e-3, random_state=42, max_iter=5, tol=None)),
               ])
sgd.fit(X_train, y_train)
```

Out[15]:
```
Pipeline(memory=None,
     steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip...dom_state=42, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False))])
```

In [16]:
```
%%time

y_pred = sgd.predict(X_test)

print('accuracy %s' % accuracy_score(y_pred, y_test))
print(classification_report(y_test, y_pred,target_names=my_tags))
```

```
accuracy 0.7894166666666667
                precision    recall  f1-score   support

          java       0.72      0.68      0.70       613
          html       0.85      0.93      0.89       620
       asp.net       0.89      0.95      0.92       587
            c#       0.81      0.80      0.80       586
ruby-on-rails       0.74      0.88      0.81       599
        jquery       0.81      0.39      0.53       589
         mysql       0.82      0.69      0.75       594
           php       0.70      0.95      0.81       610
           ios       0.83      0.55      0.66       617
    javascript       0.73      0.57      0.64       587
        python       0.70      0.67      0.68       611
             c       0.79      0.88      0.83       594
           css       0.77      0.79      0.78       619
       android       0.82      0.86      0.84       574
        iphone       0.82      0.80      0.81       584
           sql       0.72      0.68      0.70       578
  objective-c       0.81      0.90      0.85       591
           c++       0.84      0.96      0.89       608
     angularjs       0.87      0.96      0.91       638
          .net       0.78      0.88      0.83       601

     micro avg       0.79      0.79      0.79     12000
     macro avg       0.79      0.79      0.78     12000
  weighted avg       0.79      0.79      0.78     12000

Wall time: 1.41 s
```

We achieve a higher accuracy score of 79% which is 5% improvement over Naive Bayes.

Logistic Regression

In [17]:
```
from sklearn.linear_model import LogisticRegression

logreg = Pipeline([('vect', CountVectorizer()),
                   ('tfidf', TfidfTransformer()),
                   ('clf', LogisticRegression(n_jobs=1, C=1e5)),
                  ])
logreg.fit(X_train, y_train)
```

Out[17]:
```
Pipeline(memory=None,
     steps=[('vect', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip...penalty='l2', random_state=None,
          solver='warn', tol=0.0001, verbose=0, warm_start=False))])
```

In [19]:
```
%%time

y_pred = logreg.predict(X_test)

print('accuracy %s' % accuracy_score(y_pred, y_test))
print(classification_report(y_test, y_pred,target_names=my_tags))
```

```
accuracy 0.7823333333333333
                precision    recall  f1-score   support

          java       0.70      0.63      0.66       613
          html       0.91      0.91      0.91       620
       asp.net       0.97      0.94      0.95       587
            c#       0.78      0.77      0.78       586
ruby-on-rails       0.77      0.81      0.79       599
        jquery       0.59      0.58      0.59       589
         mysql       0.77      0.75      0.76       594
           php       0.82      0.86      0.84       610
           ios       0.69      0.71      0.70       617
    javascript       0.61      0.59      0.60       587
        python       0.64      0.63      0.63       611
             c       0.82      0.83      0.83       594
           css       0.78      0.77      0.77       619
       android       0.84      0.85      0.84       574
        iphone       0.80      0.83      0.82       584
           sql       0.65      0.64      0.64       578
```

objective-c	0.82	0.85	0.83	591
c++	0.91	0.91	0.91	608
angularjs	0.96	0.94	0.95	638
.net	0.78	0.83	0.80	601
micro avg	0.78	0.78	0.78	12000
macro avg	0.78	0.78	0.78	12000
weighted avg	0.78	0.78	0.78	12000

Wall time: 1.19 s

We achieve an accuracy score of 78% which is 4% higher than Naive Bayes and 1% lower than SVM.

Conclusion

We achieve an accuracy score of 78% which is 5% higher than Naive Bayes and almost 1% higher than SVM. After following some very basic steps and using a simple linear model, we were able to reach as high as an 79% accuracy on this multi-class text classification data set. Using the same data set, we can to try use some advanced techniques such as word embedding and neural networks.

References

[1] Jean Claude Utazirubanda, Tomás M. León, Papa Ngom. (2019) Variable selection with group LASSO approach: Application to Cox regression with frailty model. Communications in Statistics - Simulation and Computation 0:0, pages 1-21.

[2] Ross P. Hilton, Yuchen Zheng, Nicoleta Serban. (2018) Modeling Heterogeneity in Healthcare Utilization Using Massive Medical Claims Data. Journal of the American Statistical Association 113:521, pages 111-121.

[3] Daniela Conrad, Sarah Wilker, Anett Pfeiffer, Birke Lingenfelder, Tracie Ebalu, Hartmut Lanzinger, Thomas Elbert, Iris-Tatjana Kolassa, Stephan Kolassa. (2017) Does trauma event type matter in the assessment of traumatic load?. European Journal of Psychotraumatology 8:1.

[4] Junhui Wang, Xiaotong Shen, Yiwen Sun, Annie Qu. (2016) Classification With Unstructured Predictors and an Application to Sentiment Analysis. Journal of the American Statistical Association 111:515, pages 1242-1253.

[5] Margaret E. Roberts, Brandon M. Stewart, Edoardo M. Airoldi. (2016) A Model of Text for Experimentation in the Social Sciences. Journal of the American Statistical Association 111:515, pages 988-1003.

[6] Andrew L. Beam, Sujit K. Ghosh, Jon Doyle. (2016) Fast Hamiltonian Monte Carlo Using GPU Computing. Journal of Computational and Graphical Statistics 25:2, pages 536-548.

[7] Yun Yang, David B. Dunson. (2016) Bayesian Conditional Tensor Factorizations for High-Dimensional Classification. Journal of the American Statistical Association 111:514, pages 656-669.

[8] Na Zou, Yun Zhu, Ji Zhu, Mustafa Baydogan, Wei Wang, Jing Li. (2015) A Transfer Learning Approach for Predictive Modeling of Degenerate Biological Systems. Technometrics 57:3, pages 362-373.

[9] D.M. Sakate, D.N. Kashid. (2014) Variable selection via penalized minimum φ-divergence estimation in logistic regression. Journal of Applied Statistics 41:6, pages 1233-1246.

[10] Indrajit Mandal, N. Sairam. (2014) New machine-learning algorithms for prediction of Parkinson's disease. International Journal of Systems Science 45:3, pages 647-666.

[11] Eric C. Chi, David W. Scott. (2014) Robust Parametric Classification and Variable Selection by a Minimum Distance Criterion. Journal of Computational and Graphical Statistics 23:1, pages 111-128.

[12] Xinxing Zhou, Kaibo Wang, Lei Zhao. (2014) Variable-Selection-Based Epidemic Disease Diagnosis. Communications in Statistics - Simulation and Computation 43:7, pages 1595-1610.

[13] Wenhua Jiang, Cun-Hui Zhang. (2014) Paths Following Algorithm for Penalized Logistic Regression Using SCAD and MCP. Communications in Statistics - Simulation and Computation 43:5, pages 1064-1077.

[14] Matt Taddy. (2013) Multinomial Inverse Regression for Text Analysis. Journal of the American Statistical Association 108:503, pages 755-770.

[15] Yi Yang, Hui Zou. (2013) An Efficient Algorithm for Computing the HHSVM and Its Generalizations. Journal of Computational and Graphical Statistics 22:2, pages 396-415.