In [63]:

```
import nltk
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, Tf
idfTransformer
from sklearn import model_selection, preprocessing, linear_model, naive_bayes, m
etrics, svm
from sklearn import decomposition, ensemble
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn.neural_network import MLPClassifier
```

Base de dados

In [2]:

```
def open_file(filepath):
    file = open(filepath, 'r')

    return file.read()
```

In [3]:

```
senhora = open_file('books/senhora.txt')
diva = open_file('books/diva.txt')
gaucho = open_file('books/gaucho.txt')
guarani = open_file('books/guarani.txt')
iracema = open_file('books/iracema.txt')
luciola = open_file('books/luciola.txt')
viuvinha = open_file('books/viuvinha.txt')
ubirajara = open_file('books/ubirajara.txt')
```

In [4]:

```
# Lista com os textos crus
texts = [senhora, diva, gaucho, guarani, iracema, luciola, viuvinha, ubirajara]
```

In [5]:

```
labels = ['u', 'u', 'r', 'i', 'i', 'u', 'u', 'i']
```

In [6]:

```
collection = nltk.text.TextCollection(texts)
```

In [8]:

```
trainDF = pd.DataFrame()
trainDF['text'] = texts
trainDF['label'] = labels
trainDF
```

Out[8]:

	text	label
0	Há anos raiou no céu fluminense uma nova estre...	u
1	Emília tinha quatorze anos quando a vi pela pr...	u
2	Como são melancólicas e solenes, ao pino do so...	r
3	De um dos cabeços da Serra dos Órgãos desliza ...	i
4	Verdes mares bravios de minha terra natal, ond...	i
5	A senhora estranhou, na última vez que estivem...	u
6	Se passasse há dez anos pela Praia da Glória, ...	u
7	\n\nPela marjem do grande rio caminha Jaguarê,...	i

Vetores

In [9]:

```
count_vectorizer = CountVectorizer(ngram_range=(2,2))
tf_idf_vectorizer = TfidfVectorizer(ngram_range=(2,2))
vectorizer = TfidfVectorizer(ngram_range=(2,2), norm=None)
```

In [11]:

```
X1 = count_vectorizer.fit_transform(trainDF['text'])
X2 = tf_idf_vectorizer.fit_transform(trainDF['text'])
X3 = vectorizer.fit_transform(trainDF['text'])
```

In [14]:

```
X1.toarray()
```

Out[14]:

```
array([[0, 0, 1, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [2, 1, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

In [15]:

```
X2.toarray()
```

Out[15]:

```
array([[0.        , 0.        , 0.00247954, ..., 0.        , 0.
   ,
        0.00247954],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
   ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
   ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 0.
   ,
        0.        ],
       [0.01449272, 0.00724636, 0.        , ..., 0.        , 0.
   ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
   ,
        0.        ]])
```

In [16]:

```
X3.toarray()
```

Out[16]:

```
array([[0.        , 0.        , 2.5040774 , ..., 0.        , 0.
   ,
        2.5040774 ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
   ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
   ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 0.
   ,
        0.        ],
       [5.00815479, 2.5040774 , 0.        , ..., 0.        , 0.
   ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.
   ,
        0.        ]])
```

Classificação com Scikit-learn

In [64]:

```
classifiers = [
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    SGDClassifier(loss='hinge', penalty='l2',alpha=1e-3, n_iter=5, random_state=
42),
    DecisionTreeClassifier(max_depth=5),
    MultinomialNB(),
    GaussianNB(),
    MLPClassifier(alpha=1)]
```

In [20]:

```
train_x, test_x, train_y, test_y = model_selection.train_test_split(trainDF['tex
t'], trainDF['label'])
```

In [21]:

```
X_train = tf_idf_vectorizer.fit_transform(train_x)
```

In [22]:

```
X_test = tf_idf_vectorizer.transform(test_x)
```

In [87]:

```
def classification(clf):
    c = clf.fit(X_train, train_y)

    return c.predict(X_test)
```

In [88]:

```
svm_linear = classification(classifiers[0])
svm = classification(classifiers[1])
svm_sgdc = classification(classifiers[2])
decision_tree = classification(classifiers[3])
multi_naive = classification(classifiers[4])
neural_net = classification(classifiers[6])
```

```
/home/ellen/anaconda3/lib/python3.7/site-packages/sklearn/linear_mod
el/stochastic_gradient.py:117: DeprecationWarning: n_iter parameter
is deprecated in 0.19 and will be removed in 0.21. Use max_iter and
tol instead.
  DeprecationWarning)
```

In [78]:

```
clf = classifiers[5].fit(X_train.toarray(), train_y)
gauss_naive = clf.predict(X_test.toarray())
```

In [ ]:

```
names = ["Linear SVM", "RBF SVM", "SGDC SVM", "Decision Tree",
         "Multinominal Naive", "Naive Bayes", "Neural Net"]
```

In [89]:

```
print("Linear SVM")
print(svm_linear, np.mean(svm_linear == test_y))
print("RBF SVM")
print(svm, np.mean(svm == test_y))
print("SGDC SVM")
print(svm_sgdc, np.mean(svm_sgdc == test_y))
print("Decision Tree")
print(decision_tree, np.mean(decision_tree == test_y))
print("Multinominal Naive")
print(multi_naive, np.mean(multi_naive == test_y))
print("Gaussian Naive Bayes")
print(gauss_naive, np.mean(gauss_naive == test_y))
print("Neural Net")
print(neural_net, np.mean(neural_net == test_y))
```

```
Linear SVM
['i' 'i'] 0.0
RBF SVM
['i' 'i'] 0.0
SGDC SVM
['u' 'u'] 1.0
Decision Tree
['i' 'i'] 0.0
Multinominal Naive
['i' 'i'] 0.0
Gaussian Naive Bayes
['i' 'u'] 0.5
Neural Net
['i' 'u'] 0.5
```

Classificação com NLTK

In [90]:

```
def pre_process(raw):
    stopwords = nltk.corpus.stopwords.words('portuguese')

    tokens = nltk.word_tokenize(raw.lower())
    filtered = [t for t in tokens if t not in stopwords and t.isalpha() and len(
t) > 1]
    text = nltk.Text(tokens)

    return tokens, filtered, text
```

In [91]:

```
sra_tokens, sra_filtered, sra_text = pre_process(senhora)
diva_tokens, diva_filtered, diva_text = pre_process(diva)
gau_tokens, gau_filtered, gau_text = pre_process(gaucho)
gua_tokens, gua_filtered, gua_text = pre_process(guarani)
ira_tokens, ira_filtered, ira_text = pre_process(iracema)
luci_tokens, luci_filtered, luci_text = pre_process(luciola)
viu_tokens, viu_filtered, viu_text = pre_process(viuvinha)
ubi_tokens, ubi_filtered, ubi_text = pre_process(ubirajara)
```

In [214]:

```
tokens = [sra_filtered, diva_filtered, gau_filtered, gua_filtered, ira_filtered,
 luci_filtered, viu_filtered, ubi_filtered]
```

In [29]:

```
def TFIDF(document):
    word_tfidf = []
    for word in set(collection):
        word_tfidf.append(collection.tf_idf(word,document))
    return word_tfidf
```

In [93]:

```
collection = nltk.text.TextCollection(texts)
```

In [125]:

```
import os
from nltk.corpus.reader.plaintext import PlaintextCorpusReader

corpusdir = 'books/'
corpus = PlaintextCorpusReader(corpusdir, '.*')
```

In [123]:

```
labeled_tokens = list(zip(tokens, labels))
```

In [154]:

```
stopwords = nltk.corpus.stopwords.words('portuguese')
filtered_words = [w.lower() for w in corpus.words() if w not in stopwords and w.
isalpha() and len(w) > 1]

all_words = nltk.FreqDist(filtered_words)
```

In [155]:

```
word_features = list(all_words)[:2000]
```

In [158]:

```
def document_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    return features
```

In [136]:

```
documents = [(list(corpus.words(fileid)), category)
             for category in labels
             for fileid in corpus.fileids()]
```

In [160]:

```
featuresets = [(document_features(d), c) for (d,c) in documents]
```

In [257]:

```
train_set, test_set = featuresets[50:], featuresets[:50]
```

In [258]:

```
naivebayes = nltk.NaiveBayesClassifier.train(train_set)
```

In [259]:

```
nltk.classify.accuracy(naivebayes, test_set)
```

Out[259]:

0.38

In [260]:

```
decisiontree = nltk.DecisionTreeClassifier.train(train_set)
```

In [261]:

```
nltk.classify.accuracy(decisiontree, test_set)
```

Out[261]:

0.32

Classificação com NLTK e Scikitlearn

In [262]:

```
from nltk.classify.scikitlearn import SklearnClassifier
from sklearn.naive_bayes import MultinomialNB,BernoulliNB
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC

classifier = nltk.NaiveBayesClassifier.train(train_set)

print("Original Naive Bayes Algo accuracy percent:", (nltk.classify.accuracy(cla
ssifier, test_set))*100)
classifier.show_most_informative_features(15)
```

```
Original Naive Bayes Algo accuracy percent: 38.0
Most Informative Features
    contains(galanteria) = True                i : u      =      1.8
: 1.0
contains(particularidades) = True                i : u      =
1.8 : 1.0
   contains(esplendores) = True                i : u      =      1.8
: 1.0
       contains(frouxos) = True                i : u      =      1.8
: 1.0
         contains(bordo) = True                i : u      =      1.8
: 1.0
          contains(tato) = True                i : u      =      1.8
: 1.0
      contains(lampejos) = True                i : u      =      1.8
: 1.0
         contains(evitá) = True                i : u      =      1.8
: 1.0
         contains(sofri) = True                i : u      =      1.8
: 1.0
     contains(freqüente) = True                i : u      =      1.8
: 1.0
       contains(relevos) = True                i : u      =      1.8
: 1.0
       contains(andavam) = True                i : u      =      1.8
: 1.0
       contains(carecia) = True                i : u      =      1.8
: 1.0
      contains(saltaram) = True                i : u      =      1.8
: 1.0
        contains(físico) = True                i : u      =      1.8
: 1.0
```

In [263]:

```
MNB_classifier = SklearnClassifier(MultinomialNB())
MNB_classifier.train(train_set)
print("MNB_classifier accuracy percent:", (nltk.classify.accuracy(MNB_classifier
, test_set))*100)
```

```
MNB_classifier accuracy percent: 44.0
```

In [264]:

```
BernoulliNB_classifier = SklearnClassifier(BernoulliNB())
BernoulliNB_classifier.train(train_set)
print("BernoulliNB_classifier accuracy percent:", (nltk.classify.accuracy(Bernou
lliNB_classifier, test_set))*100)
```

```
BernoulliNB_classifier accuracy percent: 38.0
```

In [265]:

```
LogisticRegression_classifier = SklearnClassifier(LogisticRegression())
LogisticRegression_classifier.train(train_set)
print("LogisticRegression_classifier accuracy percent:", (nltk.classify.accuracy
(LogisticRegression_classifier, test_set))*100)
```

```
LogisticRegression_classifier accuracy percent: 32.0
```

In [266]:

```
SGDClassifier_classifier = SklearnClassifier(SGDClassifier())
SGDClassifier_classifier.train(train_set)
print("SGDClassifier_classifier accuracy percent:", (nltk.classify.accuracy(SGDC
lassifier_classifier, test_set))*100)
```

```
SGDClassifier_classifier accuracy percent: 52.0

/home/ellen/anaconda3/lib/python3.7/site-packages/sklearn/linear_mod
el/stochastic_gradient.py:128: FutureWarning: max_iter and tol param
eters have been added in <class 'sklearn.linear_model.stochastic_gra
dient.SGDClassifier'> in 0.19. If both are left unset, they default
to max_iter=5 and tol=None. If tol is not None, max_iter defaults to
max_iter=1000. From 0.21, default max_iter will be 1000, and default
tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

In [267]:

```
SVC_classifier = SklearnClassifier(SVC())
SVC_classifier.train(train_set)
print("SVC_classifier accuracy percent:", (nltk.classify.accuracy(SVC_classifier
, test_set))*100)
```

```
SVC_classifier accuracy percent: 32.0
```

In [268]:

```
LinearSVC_classifier = SklearnClassifier(LinearSVC())
LinearSVC_classifier.train(train_set)
print("LinearSVC_classifier accuracy percent:", (nltk.classify.accuracy(LinearSV
C_classifier, test_set))*100)
```

```
LinearSVC_classifier accuracy percent: 32.0
```

In [ ]:

```

```