Language Classification Problem

For this problem first lets check data.

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import csv
```

In [2]:
```
english_data = pd.read_csv("english.csv")
turkish_data = pd.read_csv("turkish.csv")
```

In [3]:
```
english_data.head()
```

Out[3]:

	Word
0	aah
1	aahed
2	aahing
3	aahs
4	aal

In [4]:
```
turkish_data.head()
```

Out[4]:

	Word
0	etmek
1	olmak
2	otu
3	su
4	bilimi

When I looked the head of data I detect some of abbreviations for english dataset.After checking full of data I detect many more abbreviations. In English dataset I saw suffix like 'ly','s',And as for turkish data We can't see verb suffix as "yor" "dı" "tı" but we can see "mak" for verbs. After these observations I think first we do preprocessing for this data like stemming and removing punctuation(there is not much of them but they are exist) one of the way to approuch this kind of data is character N-gram. But before we generating model lets do some preprocess.

In [5]:
```
english_data['Language'] = 0
turkish_data['Language'] = 1
```

I labeled data for future uses. In example below we can see punctuation in turkish data lets remove them. And if the word consists only of punctuations we remove that row.

In [6]:
```
turkish_data[1155:1158]
```

Out[6]:

	Word	Language
1155	...	1
1156	gömlek	1
1157	sebze	1

In [7]:

```
import string,nltk
def remove_punct(text):
    text_without_punct = "".join([char for char in text if char not in string.punctuat
    return text_without_punct
english_data['Word']=english_data['Word'].apply(lambda x: remove_punct(x))
turkish_data['Word']=turkish_data['Word'].apply(lambda x: remove_punct(x))
```

In [8]:

```
turkish_data[1155:1158]
```

Out[8]:

	Word	Language
1155		1
1156	gömlek	1
1157	sebze	1

As we can see row 1155 was "..." and became "" Lets remove this kind of rows.

In [9]:

```
turkish_data.replace("", float("NaN"), inplace=True)
turkish_data.dropna(inplace=True)
turkish_data[1155:1158]
```

Out[9]:

	Word	Language
1156	gömlek	1
1157	sebze	1
1158	karga	1

In [10]:

```
ps=nltk.PorterStemmer()
def stemming(text):
    stemmed_text = [ps.stem(word) for word in text]
    return stemmed_text
english_data['stemmed_words']= english_data['Word'].apply(lambda x: stemming([x]))
```

In [11]:

```
english_data[71:75]
```

Out[11]:

	Word	Language	stemmed_words
71	abalienate	0	[abalien]
72	abalienated	0	[abalien]
73	abalienating	0	[abalien]
74	abalienation	0	[abalien]

As we can see stemming can effect actual context(abalienation's meaning is not close to abalien)

so I will try both approuch(with stemming and without stemming).First try without stemming.(I didnt try both approuch because first one gave good results I should try it anyway but because of short amount of time I have, I didn't try it.)

In [12]:
```
all_data = turkish_data.append(english_data, ignore_index=True)
all_data.head()
```

Out[12]:

	Word	Language	stemmed_words
0	etmek	1	NaN
1	olmak	1	NaN
2	otu	1	NaN
3	su	1	NaN
4	bilimi	1	NaN

In [13]:
```
all_data.drop(['stemmed_words'], axis=1)
```

Out[13]:

	Word	Language
0	etmek	1
1	olmak	1
2	otu	1
3	su	1
4	bilimi	1
...	...	...
422079	zwinglianism	0
422080	zwinglianist	0
422081	zwitter	0
422082	zwitterion	0
422083	zwitterionic	0

422084 rows × 2 columns

In [14]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(all_data['Word'],all_data['Language']
```

Combining data, splitting it to train and test. And checking shapes to make sure everything is okay.

In [15]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(337667,)
(84417,)
```

Lineer Regresion Model Building

For model building I choice simplest approuch to begin with and before we create model we should vectorize our words with character n-gram algorithm.

In [16]:
```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(ngram_range=(3, 3),analyzer='char')
vectorizer.fit(X_train)
```

Out[16]:
```
CountVectorizer(analyzer='char', ngram_range=(3, 3))
```

In [17]:
```
print(vectorizer.transform(['bağırsağı']))
```

```
  (0, 706)      2
  (0, 760)      1
  (0, 7945)     1
  (0, 8187)     1
  (0, 12026)    1
  (0, 12201)    1
```

In [18]:
```
print(vectorizer.get_feature_names()[8187])
```

```
sağ
```

I checked if it works with simple exapmle at above

In [19]:
```
X_train = vectorizer.transform(X_train)
X_test  = vectorizer.transform(X_test)
```

In [20]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(max_iter=10000)
classifier.fit(X_train, y_train)
```

Out[20]:
```
LogisticRegression(max_iter=10000)
```

In [21]:
```
score = classifier.score(X_test, y_test)
print("Accuracy:", score)
```

```
Accuracy: 0.9741876636222562
```

In [22]:
```
X_turkish_train,X_turkish_test,y_turkish_train,y_turkish_test = train_test_split(turki
```

In [23]:
```
X_turkish_test=vectorizer.transform(X_turkish_test)
score = classifier.score(X_turkish_test, y_turkish_test)
print("Accuracy:", score)
```

```
Accuracy: 0.8600470588235294
```

In [24]:
```
test_words = ['hovarda','haydaaa','hello','harika','helyum','helium']
for word in test_words:
    if(classifier.predict(vectorizer.transform([word]))<1):
        print(word + ": English")
    else:
        print(word + ": Turkish")
```

```
hovarda: Turkish
haydaaa: Turkish
```

```
hello: English
harika: Turkish
helyum: Turkish
helium: English
```

Comments

As we can see accuracy 0.974 but because of inbalance between datasets(turkish dataset have 53k english dataset have 368k data) for turkish words accuracy is close to 0.86 we can do some processing to increase this accuracy like : increasing weight of turkish words or maybe resample with different ratio or adding some rules. Ofcourse we can create neural network with multiple layer like RNN or Lstm to making it understand weights itself and getting better result with turkish words. I used 3-gram because of RAM limitage on Jupyter Lab(2Gb) but maybe 1-gram 2-gram or combination of 3 of them can give better results we can try it in a better enviroment(More RAM) just chaging by ngram_range.

/////

From here on out I saved my model and vectorizer to load them later and use them in API

Saving Model

In [25]:
```
import pickle
model_path = "models/model.pickle"
vectorizer_path = "models/vectorizer.pickle"
pickle.dump(classifier, open(model_path, 'wb'))
pickle.dump(vectorizer, open(vectorizer_path, "wb"))
```

Loading Model

Loading must work before everything else(above) because we already have vectorizer and model at above so if we try to load them again with pickle it will give us error(To don't get this error restart kernel then come this section and run it)

In [1]:
```
import pickle
model_path = "models/model.pickle"
vectorizer_path = "models/vectorizer.pickle"

vectorizer = pickle.load(open(vectorizer_path,'rb'))
classifier = pickle.load(open(model_path,'rb'))
test_words = ['caz','blu','hell','karizmatik','ceku','müsamaha']
for word in test_words:
    if(classifier.predict(vectorizer.transform([word]))<1):
        print("English")
    else:
        print("Turkish")
```

```
Turkish
English
English
Turkish
English
Turkish
```

Creating API

In here I created a basic API which get parameter Word(example: Word:'hello') that can be called by http://127.0.0.1:5000/predict?Word=example I don't know how to create working example in Jupyter Notebook for API but following code works fine in other enviroments. I didn't create API to get multiple prediction by sending multiple word because in assigment it say "Word" only so I though I should create very simple API.

In [ ]:

```
import requests
import json
from flask import Flask,request, jsonify

app = Flask(__name__)
classifier_labels=["English","Turkish"]
@app.route('/predict')
def predict():
    req_word = request.args.get('Word')
    label_index = classifier.predict(vectorizer.transform([req_word]))
    label = classifier_labels[label_index[0]]
    return jsonify(status='complete', label=label)

if __name__ == '__main__':
    app.run(debug=True)
```