IN4080 - Mandatory assignment 2 2023

Part 1 - Exploring the NLTK tagger landscape

Exercise 1a: Data Split

In [105…
```
import nltk
nltk.download("brown")
nltk.download('universal_tagset')

from nltk.corpus import brown
sents = brown.tagged_sents(categories='news', tagset='universal')
```

```
[nltk_data] Downloading package brown to
[nltk_data]     C:\Users\MinaS\AppData\Roaming\nltk_data...
[nltk_data]   Package brown is already up-to-date!
[nltk_data] Downloading package universal_tagset to
[nltk_data]     C:\Users\MinaS\AppData\Roaming\nltk_data...
[nltk_data]   Package universal_tagset is already up-to-date!
```

In [107…
```
from sklearn.model_selection import train_test_split

news_train, news_val = train_test_split(sents,test_size=0.1)
```

Excercise 1b: Most Frequent Class Baseline

In [108…
```
tags = []
for sentence in news_train:
    for word, tag in sentence:
        tags.append(tag)

nltk.FreqDist(tags).max()
```

Out[108…
```
'NOUN'
```

In [109…
```
#import warnings
#warnings.filterwarnings('ignore')

default_tagger = nltk.DefaultTagger('NOUN')

# Evaluating on validation set:

print(f"The accuracy of the tagger against the gold standard is:
      {default_tagger.evaluate(news_val):.4f}")
```

```
The accuracy of the tagger against the gold standard is: 0.3020
```

```
C:\Users\MinaS\AppData\Local\Temp\ipykernel_23232\3809515081.py:8: DeprecationWarnin
g:
  Function evaluate() has been deprecated.  Use accuracy(gold)
  instead.
  print(f"The accuracy of the tagger against the gold standard is: {default_tagger.e
valuate(news_val):.4f}")
```

Excercise 1c: Naive Bayes Unigram Tagger

In [110…

```
unigram_tagger = nltk.UnigramTagger(news_train)

print(f"The accuracy of the tagger against the gold standard is:
      {unigram_tagger.accuracy(news_val):.4f}")
```

```
The accuracy of the tagger against the gold standard is: 0.8809
```

The accuracy on the universal tagset is alot higher on the unigram tagger than the default tagset as they differ as much as 0.49.

Excercise 1d: Bigram HMM Tagger

In [111…

```
hmm = nltk.HiddenMarkovModelTagger.train(news_train)
print(f"The accuracy of the tagger against the gold standard is:
      {hmm.evaluate(news_val):.4f}")
```

```
C:\Users\MinaS\AppData\Local\Temp\ipykernel_23232\3538903188.py:2: DeprecationWarnin
g:
  Function evaluate() has been deprecated.  Use accuracy(gold)
  instead.
  print(f"The accuracy of the tagger against the gold standard is: {hmm.evaluate(new
s_val):.4f}")
The accuracy of the tagger against the gold standard is: 0.9096
```

The accuracy is even higher than the unigram tagger.

1e: Perceptron with greedy decoding

In [112…

```
perc = nltk.PerceptronTagger(load=False)
perc.train(news_train)
print(f"The accuracy of the tagger against the gold standard is:
      {perc.evaluate(news_val):.4f}")
```

```
C:\Users\MinaS\AppData\Local\Temp\ipykernel_23232\454315405.py:3: DeprecationWarnin
g:
  Function evaluate() has been deprecated.  Use accuracy(gold)
  instead.
  print(f"The accuracy of the tagger against the gold standard is: {perc.evaluate(ne
ws_val):.4f}")
The accuracy of the tagger against the gold standard is: 0.9657
```

This algorithm has better accuracy than all the previous.

Part 2 - Greedy LR taggers and feature engineering

Exercise 2a: Getting started with a greedy logistic regression tagger

In [113…

```
import nltk
import numpy as np
import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction import DictVectorizer


class ScikitGreedyTagger(nltk.TaggerI):
    def __init__(self, features, clf=LogisticRegression(max_iter=1000)):
        self.features = features
        self.classifier = clf
        self.vectorizer = DictVectorizer()

    def train(self, train_sents):
        train_feature_sets = []
        train_labels = []

        for tagged_sent in train_sents:
            history = []
            untagged_sent = nltk.tag.untag(tagged_sent)

            for i, (word, tag) in enumerate(tagged_sent):
                feature_set = self.features(untagged_sent, i, history)
                train_feature_sets.append(feature_set)
                train_labels.append(tag)
                history.append(tag)

        x_train = self.vectorizer.fit_transform(train_feature_sets)
        y_train = np.array(train_labels)
        self.classifier.fit(x_train, y_train)

    def tag(self, sentence):
        history = []
        for i, word in enumerate(sentence):
            featureset = self.features(sentence, i, history)
            X_test = self.vectorizer.transform(featureset)
            tags = self.classifier.predict(X_test)
            history.append(tags)
        return zip(sentence, history)
```

In [114…

```
def pos_features(sentence, i, history):
    features = {"curr_word": sentence[i]}
    if i == 0:
        features["prev_word"] = "<START>"
    else:
        features["prev_word"] = sentence[i-1]
    return features
```

In [115…

```
lr_tagger = ScikitGreedyTagger(pos_features)
lr_tagger.train(news_train)
lr_tagger.accuracy(news_val)
```

Out[115…

```
0.920281483037496
```

The accuracy of this tagger compared to the DaultTagger, UnigramTagger and HiddenMarkovModelTagger is higher but lower copared to PerceptronTagger.

Exercise 2b: Adding word context features

In [116…

```
def pos_features_2b(sentence, i, history):
    features = {"curr_word": sentence[i]}
    if i == 0:
        features["prev_prev_word"] = "<START 2>"
        features["prev_word"] = "<START>"
    elif i == 1:
        features["prev_prev_word"] = "<START>"
        features["prev_word"] = sentence[i-1]
    else:
        features["prev_prev_word"] = sentence[i-2]
        features["prev_word"] = sentence[i-1]

    if i == (len(sentence)-1):
        features["next word"] = "<END>"
    else:
        features["next word"] = sentence[i+1]

    return features
```

In [117…

```
lr_tagger = ScikitGreedyTagger(pos_features_2b)
lr_tagger.train(news_train)
lr_tagger.accuracy(news_val)
```

Out[117…

```
0.9289990547211427
```

The combination that works best is the last one which added the next word and the word before the previous one as its accuracy was higher. However greedy decoding still has the highest accuracy compared to all of the taggers.

Excercise 2c: Adding transition features

In [118…

```
def pos_features_2c_digram(sentence, i, history):
    features = {"curr_word": sentence[i]}
    if i == 0:
        features["prev_word"] = "<START>"
        features["prev_tag"] = "<START>"
    else:
        features["prev_word"] = sentence[i-1]
        features["prev_tag"] = history[i-1]
    return features
```

In [119…

```
lr_tagger = ScikitGreedyTagger(pos_features_2c_digram)
lr_tagger.train(news_train)
lr_tagger.accuracy(news_val)
```

Out[119…

```
0.9165003676084444
```

In [120…

```
def pos_features_2c_trigram(sentence, i, history):
    features = {}
    if i == 0:
        features["prev_prev_word"] = "<START>"
        features["prev_word"] = "<START>"
    elif i == 1:
        features["prev_prev_word"] = "<START>"
        features["prev_word"] = sentence[i-1]
        features["prev_tag"] = history[i-1]
    else:
        features["prev_prev_word"] = sentence[i-2]
        features["prev_prev_tag"] = history[i-2]
        features["prev_word"] = sentence[i-1]
        features["prev_tag"] = history[i-1]
    features["curr_word"] = sentence[i]

    if i == (len(sentence)-1):
        features["next word"] = "<END>"
    else:
        features["next word"] = sentence[i+1]

    return features
```

In [122…

```
lr_tagger = ScikitGreedyTagger(pos_features_2c_trigram)
lr_tagger.train(news_train)
lr_tagger.accuracy(news_val)
```

Out[122…

```
0.9285789307845814
```

Excersice 2d: Even more features

Try to add more features to get an even better tagger. Only the fantasy sets limits to what you may consider. Some ideas: Extract suffixes and prefixes from the current, previous or next word. Is the current word a number? Is it capitalized? Does it contain capitals? Does it contain a hyphen? etc. What is the best feature set you can come up with? Train and test various feature sets and select the best one.

If you use sources for finding tips about good features (like articles, web pages, NLTK code, etc.) make references to the sources and explain what you got from them.

In [123…

```
import re
```

In [124…

```
def pos_features_2d_trigram_suf(sentence, i, history):
    features = {"suffix(1)": sentence[i][-1:],
                "suffix(2)": sentence[i][-2:],
                "suffix(3)": sentence[i][-3:],
                "prefix(3)": sentence[i][:3]
                }
    if i == 0:
        features["prev_prev_word"] = "<START>"
        features["prev_word"] = "<START>"
    elif i == 1:
        features["prev_prev_word"] = "<START>"
        features["prev_word"] = sentence[i-1]
        features["prev_prev_tag"] = "<START>"
        features["prev_tag"] = history[i-1]
    else:
        features["prev_prev_word"] = sentence[i-2]
        features["prev_prev_tag"] = history[i-2]
        features["prev_word"] = sentence[i-1]
        features["prev_tag"] = history[i-1]

    features["curr_word"] = sentence[i]

    if i == (len(sentence)-1):
        features["next word"] = "<END>"
    else:
        features["next word"] = sentence[i+1]

    return features
```

In [125…

```
lr_tagger = ScikitGreedyTagger(pos_features_2d_trigram_suf)
lr_tagger.train(news_train)
lr_tagger.accuracy(news_val)
```

Out[125…

```
0.9633441865350278
```

In [126…

```
def pos_features_2d_trigram_num(sentence, i, history):
    features = {"suffix(1)": sentence[i][-1:],
                "suffix(2)": sentence[i][-2:],
                "suffix(3)": sentence[i][-3:],
                "prefix(3)": sentence[i][:3],
                "Numeric": sentence[i].isdigit()
                }
    if i == 0:
        features["prev_prev_word"] = "<START>"
        features["prev_word"] = "<START>"
    elif i == 1:
        features["prev_prev_word"] = "<START>"
        features["prev_word"] = sentence[i-1]
        features["prev_prev_tag"] = "<START>"
```

```python
        features["prev_tag"] = history[i-1]
    else:
        features["prev_prev_word"] = sentence[i-2]
        features["prev_prev_tag"] = history[i-2]
        features["prev_word"] = sentence[i-1]
        features["prev_tag"] = history[i-1]

    features["curr_word"] = sentence[i]

    if i == (len(sentence)-1):
        features["next word"] = "<END>"
    else:
        features["next word"] = sentence[i+1]

    return features
```

In [127…

```python
lr_tagger = ScikitGreedyTagger(pos_features_2d_trigram_num)
lr_tagger.train(news_train)
lr_tagger.accuracy(news_val)
```

Out[127…

```
0.9633441865350278
```

In [128…

```python
def pos_features_2d_trigram_cap(sentence, i, history):
    features = {"suffix(1)": sentence[i][-1:],
                "suffix(2)": sentence[i][-2:],
                "suffix(3)": sentence[i][-3:],
                "prefix(3)": sentence[i][:3],
                "Capitalized": sentence[i].isupper(),
                "Any_uppercase": any(ele.isupper() for ele in sentence[i])
                }
    if i == 0:
        features["prev_prev_word"] = "<START>"
        features["prev_word"] = "<START>"
    elif i == 1:
        features["prev_prev_word"] = "<START>"
        features["prev_word"] = sentence[i-1]
        features["prev_prev_tag"] = "<START>"
        features["prev_tag"] = history[i-1]
    else:
        features["prev_prev_word"] = sentence[i-2]
        features["prev_prev_tag"] = history[i-2]
        features["prev_word"] = sentence[i-1]
        features["prev_tag"] = history[i-1]

    features["curr_word"] = sentence[i]

    if i == (len(sentence)-1):
        features["next word"] = "<END>"
    else:
        features["next word"] = sentence[i+1]

    return features
```

In [129…

```python
lr_tagger = ScikitGreedyTagger(pos_features_2d_trigram_cap)
lr_tagger.train(news_train)
```

```
lr_tagger.accuracy(news_val)
```

Out[129… 0.9641844344081504

In [130…
```
def pos_features_2d_trigram_hyp(sentence, i, history):
    features = {"suffix(1)": sentence[i][-1:],
                "suffix(2)": sentence[i][-2:],
                "suffix(3)": sentence[i][-3:],
                "prefix(3)": sentence[i][:3],
                "Capitalized": sentence[i].isupper(),
                "Any_uppercase": any(ele.isupper() for ele in sentence[i]),
                "Hyphen":  bool(re.search("-", sentence[i]))
                }
    if i == 0:
        features["prev_prev_word"] = "<START>"
        features["prev_word"] = "<START>"
    elif i == 1:
        features["prev_prev_word"] = "<START>"
        features["prev_word"] = sentence[i-1]
        features["prev_prev_tag"] = "<START>"
        features["prev_tag"] = history[i-1]
    else:
        features["prev_prev_word"] = sentence[i-2]
        features["prev_prev_tag"] = history[i-2]
        features["prev_word"] = sentence[i-1]
        features["prev_tag"] = history[i-1]

    features["curr_word"] = sentence[i]

    if i == (len(sentence)-1):
        features["next word"] = "<END>"
    else:
        features["next word"] = sentence[i+1]

    return features
```

In [131…
```
lr_tagger = ScikitGreedyTagger(pos_features_2d_trigram_hyp)
lr_tagger.train(news_train)
lr_tagger.accuracy(news_val)
```

Out[131… 0.9654448062178342

reference: NLTK book link: https://www.nltk.org/book/ch06.html result:

There were improvment on all of the functions when adding features, but when adding the hyphen and numeric feature the accuracy dropped by 0.001.

Exercise 2e: Regularization

As in the previous assignment, we will study the effect of different regularization strengths now. In scikit-learn, regularization is expressed by the parameter C. A smaller C means stronger regularization. Try with C in [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0] and see which value which yields the best result. You can also try additional values. Summarize your experiments to make clear which set of features and parameters provide the best results, and what the corresponding accuracy score is. Did you manage to outperform the perceptron tagger? If not, where do you think the bottleneck of your current tagger lies?

Answer:

It almost outperformed the perceptron tagger with c=10. The bottleneck of current tagger lies in how big c is as it will work slower when this value is bigger.

In [132…
```
C = [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
```

In [133…
```
Max_accuracy = 0
best_c = 0
for c in C:
    cls = ScikitGreedyTagger(features=pos_features_2d_trigram_cap,clf=LogisticRegre
    cls.train(news_train)
    acc = cls.accuracy(news_val)
    if acc > Max_accuracy:
        Max_accuracy = acc
        best_c = c

print(f"The best accuracy is related to c = {best_c} and is equal to accuracy = {Ma
```

```
c:\Users\MinaS\Anaconda3\envs\in4080_2023\lib\site-packages\sklearn\linear_model\_lo
gistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
c:\Users\MinaS\Anaconda3\envs\in4080_2023\lib\site-packages\sklearn\linear_model\_lo
gistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
c:\Users\MinaS\Anaconda3\envs\in4080_2023\lib\site-packages\sklearn\linear_model\_lo
gistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
c:\Users\MinaS\Anaconda3\envs\in4080_2023\lib\site-packages\sklearn\linear_model\_lo
gistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
c:\Users\MinaS\Anaconda3\envs\in4080_2023\lib\site-packages\sklearn\linear_model\_lo
gistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
The best accuracy is related to c = 10.0 and is equal to accuracy = 0.96670517802751
81
```

Part 3 - Training and testing on a larger corpus

Exercise 3a: Compile the extended training and test data

The NLTK book, chapter 2.1.3, lists the names of the 15 genres available in the Brown corpus. We will set two genres aside for testing: hobbies and adventure. For training, we will use the news training set prepared for the previous exercises, as well as the data from the remaining 12 genres. Prepare the corpus as described and store the datasets in the variables all_train, hobbies_test and adventure_test. We will not use news_val in this part. Make sure to use the universal tagset.

In [134…

```
categories = brown.categories()

hobbies_test = brown.tagged_sents(categories='hobbies', tagset='universal')
adventure_test = brown.tagged_sents(categories='adventure', tagset='universal')

train_categories = []
for category in categories:
    if category not in ['hobbies', 'adventure', 'news']:
        train_categories.append(category)

train = brown.tagged_sents(categories=train_categories, tagset='universal')
```

In [135…

```
all_train = train + news_train
```

3b: Evaluate the taggers

Identify the most successful tagger from part 1 and the best setup from part 2. Retrain both of them on all_train and evaluate them separately on the two test genres. Report the results and discuss them briefly: Which of the two genres is “easier”? How well do the two taggers generalize to unseen genres?

In [136…

```
# Most successful tagger from part 1 with accuracy of 0.97
perc2 = nltk.PerceptronTagger(load=False)
perc2.train(all_train)

# Best setup from part 2 is c = 10.0 with accuracy = 0.969
clscap = ScikitGreedyTagger(features=pos_features_2d_trigram_cap,clf=LogisticRegres
clscap.train(all_train)
```

```
c:\Users\MinaS\Anaconda3\envs\in4080_2023\lib\site-packages\sklearn\linear_model\_lo
gistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [137…

```
# Testing on hobbies set
print(clscap.accuracy(hobbies_test))

# Testing on adventure set
print(clscap.accuracy(adventure_test))
```

```
0.9613577023498695
0.9697297453202965
```

In [138…

```python
# Testing on hobbies set
print(perc2.accuracy(hobbies_test))

# Testing on adventure set
print(perc2.accuracy(adventure_test))
```

```
0.9674418604651163
0.9743012892619192
```

The genre that is easier is adventure. It seems like the perceptron tagger generalises a little bit better than the other one, but in general they both are able to generalise pretty well as they have high accuracy for the unseen test data.

Excercise 3c: Confusion matrix

The accuracy gives us a high-level overview of the performance of a tagger, but we may be interested in finding out more details about where the tagger makes the mistakes. The universal tagset is reasonably small, so we can produce a confusion matrix. Take a look at https://www.nltk.org/api/nltk.tag.api.html and make a confusion matrix for the results. Pick the results of one test set and one tagger. Make sure you understand what the rows and columns are. Which pairs of tags are most easily confounded?

You can find the documentation of the tagset in the following link, but note that NLTK uses an earlier, slightly different version of the tagset: https://universaldependencies.org/u/pos/index.htm

In [139…

```python
# Perceptron hobbies set
print(perc2.confusion(hobbies_test))
```

```
     |                             C           N           P           V       |
     |           A     A     A     O     D     O     N     R     P     E       |
     |           D     D     D     N     E     U     U     O     R     R       |
     |     .     J     P     V     J     T     N     M     N     T     B     X |
-----+-------------------------------------------------------------------------+
   . | <9792>    .     .     .     .     .     .     .     .     .     .     . |
 ADJ |     . <6071>    6    95     .     .   297    19     .     2    69     . |
 ADP |     .     6 <9932>   31     3     3     6     .     6    80     3     . |
 ADV |     .   119    97 <3432>    5     9    53     .     .    15     6     . |
CONJ |     .     .     3     4 <2937>    2     2     .     .     .     .     . |
 DET |     .     .    18     5     . <9469>    .     .     5     .     .     . |
NOUN |     .   419     6    34     .     1<20427>   69     .     4   316     . |
 NUM |     .    15     .     .     .     .    35 <1375>    .     1     .     . |
PRON |     .     .    19     .     .     9     .     . <2306>    .     .     . |
 PRT |     .     1   133    11     .     .     7     .     . <1737>    .     . |
VERB |     .    62     8    13     .     .   486     .     .     .<12164>    . |
   X |     .     4     .     .     .     1    57     .     .     .     1   <22>|
-----+-------------------------------------------------------------------------+
(row = reference; col = test)
```

In [140…

```python
# Perceptron Adventure set
print(perc2.confusion(adventure_test))
```

```
     |                                       N                                       |
     |                   C                   O                   P                   |
     |         A    A    A         D         U         N         R    P    V         |
     |         D    D    D    O    E         N         U         O    R    E         |
     |    .    J    P    V    N    T         N         M         N    T    B    X    |
-----+-----------------------------------------------------------------------------+
   . |<10929>  .    .    .    .    .    .    .    .    .    .    .  |
 ADJ |    . <3040>   6  113    .    .  151    1    .    1   52    .  |
 ADP |    .    5 <6916>  41    3    .   13    .    3   83    5    .  |
 ADV |    .  114   48 <3598>   6   16   59    1    .   25   12    .  |
CONJ |    .    .    2    1 <2155>  12    2    .    .    .    1    .  |
 DET |    .    .   20   13    2 <8079>   3    2   36    .    .    .  |
NOUN |    .  159    1   35    .    .<13013>  16    1    8  121    .  |
 NUM |    .    1    .    .    .    .    6 <459>    .    .    .    .  |
PRON |    .    .    8    2    .   85    9    . <5099>   1    1    .  |
 PRT |    .    4  132   19    .    .   72    .    1 <2197>  11    .  |
VERB |    .   33   11   11    .    1  139    .    .    5<12074>   .  |
   X |    .    .    .    .    .    .   34    .    .    .    3  <1>|
-----+-----------------------------------------------------------------------------+
(row = reference; col = test)
```

In [145…

```python
from sklearn.metrics import classification_report, confusion_matrix
from itertools import chain

def reporting_accuracy(report, tagger, test_set):
    # Get predictions
    predicted_sents = tagger.tag_sents([[word for word, _ in sent] for sent in test

    # Extract tags from test data and predictions
    true_tags = list(chain.from_iterable([tag for _, tag in sent] for sent in test_
    predicted_tags = list(chain.from_iterable([tag for _, tag in sent] for sent in

    # Print classification report
    print(report(true_tags, predicted_tags))
```

In [146…

```python
# log tagger hobbies set
reporting_accuracy(confusion_matrix, clscap, hobbies_test)
```

```
[[ 9792     0     0     0     0     0     0     0     0     0     0     0]
 [    0  5817     6   195     0     3   373     1     0     2   162     0]
 [    0     7  9795    58     4    10     5     0     8   180     2     1]
 [    0   136   121  3405     4     8    40     0     0    17     5     0]
 [    0     0     6     2  2938     2     0     0     0     0     0     0]
 [    0     0     9     6     0  9478     0     0     4     0     0     0]
 [   29   350    13    63     1     5 20283    48     4    13   452    15]
 [    0     4     0     0     0     0    17  1405     0     0     0     0]
 [    0     0    15     0     0    13     0     0  2306     0     0     0]
 [    0     1    83    12     0     0     6     0     0  1787     0     0]
 [    0    76    20    17     0     2   480     1     0     2 12135     0]
 [    0     4     0     0     0     2    55     0     0     1     1    22]]
```

In [148…

```python
# log tagger adventure set
reporting_accuracy(confusion_matrix, clscap, adventure_test)
```

```
[[10929     0     0     0     0     0     0     0     0     0     0     0]
 [    0  2927     6   157     0     1   163     2     1     2   105     0]
 [    0     8  6808    61     3     7     8     0     7   159     8     0]
 [    0   113    48  3599    13    14    39     0     0    35    18     0]
 [    0     0     5     0  2155    13     0     0     0     0     0     0]
 [    0     0    14    22     2  8076     1     2    37     0     1     0]
 [    0   188     1    41     0     3 12906    19     4    22   163     7]
 [    0     0     0     0     0     0     6   459     0     0     1     0]
 [    0     1     9     0     0    90     7     0  5093     5     0     0]
 [    0     4   109    21     0     0    44     0     1  2247    10     0]
 [    0    51    10    13     0     0   150     0     1     6 12043     0]
 [    0     0     0     0     0     0    33     0     1     2     1     1]]
```

Excercise 3d: Precision, recall and f-measure

Finding hints on the NLTK web page linked above, calculate the precision, recall and f-measure for each tag and display the results in a table.

Also calculate the macro precision, macro recall and macro f-measure across all tags.

In [ ]:
```
print("Perceptron Hobbies")
print(perc2.evaluate_per_tag(hobbies_test))
print("Perceptron Advantures")
print(perc2.evaluate_per_tag(adventure_test))
```

```
Perceptron Hobbies
 Tag | Prec.  | Recall | F-measure
-----+--------+--------+-----------
   . | 1.0000 | 1.0000 | 1.0000
 ADJ | 0.9089 | 0.9290 | 0.9188
 ADP | 0.9720 | 0.9872 | 0.9796
 ADV | 0.9521 | 0.9194 | 0.9355
CONJ | 0.9973 | 0.9969 | 0.9971
 DET | 0.9978 | 0.9972 | 0.9975
NOUN | 0.9566 | 0.9611 | 0.9589
 NUM | 0.9442 | 0.9734 | 0.9586
PRON | 0.9957 | 0.9897 | 0.9927
 PRT | 0.9478 | 0.9227 | 0.9351
VERB | 0.9696 | 0.9553 | 0.9624
   X | 0.9231 | 0.2824 | 0.4324

Perceptron Advantures
 Tag | Prec.  | Recall | F-measure
-----+--------+--------+-----------
   . | 1.0000 | 1.0000 | 1.0000
 ADJ | 0.9051 | 0.9040 | 0.9045
 ADP | 0.9690 | 0.9769 | 0.9730
 ADV | 0.9377 | 0.9276 | 0.9326
CONJ | 0.9922 | 0.9913 | 0.9917
 DET | 0.9866 | 0.9913 | 0.9889
NOUN | 0.9648 | 0.9746 | 0.9697
 NUM | 0.9586 | 0.9936 | 0.9758
PRON | 0.9926 | 0.9800 | 0.9863
 PRT | 0.9445 | 0.9085 | 0.9261
VERB | 0.9838 | 0.9832 | 0.9835
   X | 1.0000 | 0.0526 | 0.1000
```

In [ ]:
```python
reporting_accuracy(classification_report, clscap, adventure_test)
```

```
              precision    recall  f1-score   support

           .       1.00      1.00      1.00     10929
         ADJ       0.89      0.87      0.88      3364
         ADP       0.97      0.97      0.97      7069
         ADV       0.93      0.92      0.92      3879
        CONJ       0.99      0.99      0.99      2173
         DET       0.98      0.99      0.99      8155
        NOUN       0.96      0.97      0.97     13354
         NUM       0.95      0.99      0.97       466
        PRON       0.99      0.98      0.98      5205
         PRT       0.91      0.91      0.91      2436
        VERB       0.98      0.98      0.98     12274
           X       0.25      0.03      0.05        38

    accuracy                           0.97     69342
   macro avg       0.90      0.88      0.88     69342
weighted avg       0.97      0.97      0.97     69342
```

In [ ]:
```python
reporting_accuracy(classification_report, clscap, hobbies_test)
```

```
              precision    recall  f1-score   support

           .       1.00      1.00      1.00      9792
         ADJ       0.91      0.89      0.90      6559
         ADP       0.97      0.98      0.97     10070
         ADV       0.92      0.90      0.91      3736
        CONJ       1.00      1.00      1.00      2948
         DET       1.00      1.00      1.00      9497
        NOUN       0.95      0.96      0.95     21276
         NUM       0.96      0.98      0.97      1426
        PRON       0.99      0.99      0.99      2334
         PRT       0.92      0.93      0.92      1889
        VERB       0.96      0.95      0.95     12733
           X       0.50      0.26      0.34        85

    accuracy                           0.96     82345
   macro avg       0.92      0.90      0.91     82345
weighted avg       0.96      0.96      0.96     82345
```

Excercise 3e: Error analysis

Sometimes, it makes sense to inspect the output of a machine learning model more thoroughly. Find five sentences in the test set where at least one token is misclassified and display these sentences in the following format, with both the predicted and gold tags.

Identify the words that are tagged differently. Comment on each of the differences. Would you say that the predicted tag is wrong? Or is there a genuine ambiguity such that both answers are defendable? Or is even the gold tag wrong?

Answer:

For the perceptron:

word------- predicted ---------- gold:

1. her------- DET---------- PRON

A determiner is a word that introduces a noun and provides information about the oun, while pronoun is a word used to replace a noun to avoid repetition.Thus there could be ambiguity between these. "Her" could be a possessive determiner, but it has to be used before nouns or noun phrases. Since here it was not used before noun phrases it is not a determiner.

2. fickle------- VERB ---------- ADJ

There could not be a ambiguity here.

3. sleep------- VERB---------- NOUN

sleep could be both verb and noun, but here it was wrongly predicted as verb.

4. her------- DET---------- PRON

Here we have the same issue er (1). Thus the tag has been predicted wrong.

4. much------- ADJ---------- ADV

Adjectives are words that describe the qualities or state of being of nouns. Adverbs are wordds that describes a verb. eks. "He sings loudly", here "loudly" is an adverb.adverbs can also describe a whole sentence eks. "Fortuenately, i had brougt an umbrella.", here "Fortunately" is adverb. Thus the tag was predicted wrongly.

5. stubborn------- NOUN---------- ADJ

wrongly predicted tag.

In [156…

```
def reporting_accuracy_sents(tagger, test_set):


    # Get predictions
    predicted_sents = tagger.tag_sents([[word for word, _ in sent] for sent in test
    sents = list([word for word,_ in sent] for sent in test_set)
    true_tags = list([tag for _, tag in sent] for sent in test_set)
    predicted_tags = list([tag for _, tag in sent] for sent in predicted_sents)

    s = 0
    for i in range(len(test_set)):
        if predicted_tags[i] != true_tags[i] and s < 5:
            s +=1
            print("\n \n")
            print("Token"+" "*15 + "pred" + " "*5 + "gold")
            print("-"*35)
            for j in range(len(predicted_tags[i])):
                word_width = 20
                tag_width = 10

                # Format and print each field with equal width
                formatted_output = "{:<{}}{:<{}}{:<{}}".format(sents[i][j], word_wi
                print(formatted_output)
```

In [157…

```
reporting_accuracy_sents(perc, adventure_test)
```

Token	pred	gold
He	PRON	PRON
was	VERB	VERB
well	ADV	ADV
rid	ADJ	ADJ
of	ADP	ADP
her	DET	PRON
.	.	.

Token	pred	gold
He	PRON	PRON
certainly	ADV	ADV
didn't	VERB	VERB
want	VERB	VERB
a	DET	DET
wife	NOUN	NOUN
who	PRON	PRON
was	VERB	VERB
fickle	VERB	ADJ
as	ADP	ADP
Ann	NOUN	NOUN
.	.	.

Token	pred	gold
Sometimes	ADV	ADV
he	PRON	PRON
woke	VERB	VERB
up	PRT	PRT
in	ADP	ADP
the	DET	DET
middle	NOUN	NOUN
of	ADP	ADP
the	DET	DET
night	NOUN	NOUN
thinking	NOUN	VERB
of	ADP	ADP
Ann	NOUN	NOUN
,	.	.
and	CONJ	CONJ
then	ADV	ADV
could	VERB	VERB
not	ADV	ADV
get	VERB	VERB
back	ADV	ADV
to	ADP	ADP
sleep	VERB	NOUN
.	.	.

Token	pred	gold
His	DET	DET
plans	NOUN	NOUN
and	CONJ	CONJ
dreams	NOUN	NOUN
had	VERB	VERB
revolved	VERB	VERB
around	ADP	ADP
her	DET	PRON
so	ADV	ADV
much	ADJ	ADV
and	CONJ	CONJ
for	ADP	ADP
so	ADV	ADV
long	ADJ	ADJ
that	ADP	ADP
now	ADV	ADV
he	PRON	PRON
felt	VERB	VERB
as	ADP	ADP
if	ADP	ADP
he	PRON	PRON
had	VERB	VERB
nothing	NOUN	NOUN
.	.	.

Token	pred	gold
The	DET	DET
easiest	ADJ	ADJ
thing	NOUN	NOUN
would	VERB	VERB
be	VERB	VERB
to	PRT	PRT
sell	VERB	VERB
out	PRT	PRT
to	ADP	ADP
Al	NOUN	NOUN
Budd	NOUN	NOUN
and	CONJ	CONJ
leave	VERB	VERB
the	DET	DET
country	NOUN	NOUN
,	.	.
but	CONJ	CONJ
there	PRT	PRT
was	VERB	VERB
a	DET	DET
stubborn	NOUN	ADJ
streak	NOUN	NOUN

in	ADP	ADP
him	PRON	PRON
that	ADP	DET
wouldn't	VERB	VERB
allow	VERB	VERB
it	PRON	PRON
.	.	.