

Εργασία 3 – Επισημείωση Ακολουθιών

Επεξεργασία Φυσικής Γλώσσας

**Τζανής Νικόλαος
mtn2217**

Ερώτημα 1

Από την εκτέλεση του κώδικα που δίνεται μάς επιστρέφονται οι εξής μετρήσεις για το accuracy και το macro-accuracy στο test dataset:

```
Test Accuracy : 0.980
```

```
Test Macro-Accuracy : 0.910
```

Ερώτημα 2

Με αυτό το κομμάτι κώδικα επιστρέφεται μία τυχαία πρόταση μεγέθους πάνω από 10 tokens για την οποία το μοντέλο μας δεν βρήκε τα σωστά tags. Ακολουθώντας εκτυπώνεται η πρόταση, ενώ στον πίνακα βλέπουμε το πραγματικό tag (actual tag) καθώς και το tag που προέβλεψε το μοντέλο μας:

```
import numpy as np

def find_sentence_with_at_least_one_wrong_label(Y_actual, Y_preds):
    for i, (preds, labels) in enumerate(zip(Y_preds, Y_actual)):
        num_of_wrong_labels = sum(np.array(preds) != np.array(labels))
        if len(labels) >= 10 and num_of_wrong_labels > 0:
            tokens = test_sentences[i]["tokens"]
            preds = preds[labels != -100]
            predictions = [tagmap.get_itos()[p] for p in preds]
            labels = labels[labels != -100]
            labels = [tagmap.get_itos()[l] for l in labels]
            break
    return tokens, predictions, labels

Y_actual, Y_preds = EvaluateModel(model, test_loader, question_2 = True)
tokens, predictions, labels = find_sentence_with_at_least_one_wrong_label(Y_actual, Y_preds)

print("\nThe actual and predicted tags for the tokens of this sentence were:")
print("\n{:<10} {:<10} {:<10}".format('Token', 'Actual Tag', 'Predicted Tag'))

for i in range(len(tokens)):
    print("{:<10} {:<10} {:<10}".format(tokens[i], labels[i], predictions[i]))
```

Το αποτέλεσμα της εκτέλεσης του παραπάνω κώδικα:

The actual and predicted tags for the tokens of this sentence were:

Token	Actual Tag	Predicted Tag
SOCCER	O	O
-	O	O
JAPAN	B-LOC	B-LOC
GET	O	O
LUCKY	O	O
WIN	O	O
,	O	O
CHINA	B-PER	B-LOC
IN	O	O
SURPRISE	O	O
DEFEAT	O	O
.	O	O

Όπως βλέπουμε, στη λέξη 'CHINA' θα έπρεπε να έχει δοθεί το B-PER tag, αλλά το μοντέλο έδωσε B-LOC. Για όλα τα υπόλοιπα tokens αυτής της πρότασης, το prediction του μοντέλου ήταν σωστό.

Ακολουθως επιλέγουμε μία δική μας πρόταση και επαναλαμβάνουμε την παραπάνω διαδικασία. Η πρόταση που έχω επιλέξει προέρχεται από το Wall Street Journal:

```
test_sentence_dict = {'tokens': ['Zurich', 'and', 'some', 'other', 'insurers', 'with', 'big', 'fleets', 'of', 'engineers', 'are', 'advising', 'companies', 'on', 'how', 'to', 'fortify', 'their', 'properties', '.'],
                    'pos_tags': ['NNP', 'CC', 'DT', 'JJ', 'NNS', 'IN', 'JJ', 'NNS', 'IN', 'NNP', 'VBP', 'VBD', 'NNS', 'IN', 'NNP', 'TO', 'VB', 'VBZ', 'NNP', '.'],
                    'chunk_tags': ['B-NP', 'O', 'B-NP', 'I-NP', 'B-NP', 'B-PP', 'B-NP', 'I-NP', 'B-PP', 'B-NP', 'B-VP', 'B-VP', 'B-NP', 'B-PP', 'B-ADV', 'B-VP', 'B-NP', 'I-NP', 'O'],
                    'ner_tags': ['B-ORG', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']}

new_test_dataset = [encode(test_sentence_dict)]
new_test_loader = torch.utils.data.DataLoader(new_test_dataset, batch_size=1)
new_y_actual, new_y_preds = evaluateModel(model, new_test_loader)

print("\nThe actual and predicted tags for the tokens of this sentence were:")
print("\n{:<10} {:<10} {:<10}".format('tokens', 'Actual Tag', 'Predicted Tag'))

for i in range(len(test_sentence_dict['tokens'])):
    token = test_sentence_dict['tokens'][i]
    print("\n{:<10} {:<10} {:<10}".format(token, tagmap.get_itos()[new_y_actual[i].item()], tagmap.get_itos()[new_y_preds[i].item()])))
```

Η πρόταση βρίσκεται ήδη στη μορφή dictionary που χρειάζεται για να δώσει predictions το μοντέλο. Το αποτέλεσμα της εκτέλεσης του παραπάνω κελιού είναι:

```
The actual and predicted tags for the tokens of this sentence were:
```

Token	Actual Tag	Predicted Tag
Zurich	B-ORG	B-ORG
and	O	O
some	O	O
other	O	O
insurers	O	O
with	O	O
big	O	O
fleets	O	O
of	O	O
engineers	O	O
are	O	O
advising	O	O
companies	O	O
on	O	O
how	O	O
to	O	O
fortify	O	O
their	O	O
properties	O	O
.	O	O

Άρα όπως βλέπουμε, το μοντέλο δεν έχει κάνει λάθος στο prediction των tags.

Ερώτημα 3

Η μέθοδος `align_label` παίρνει ως είσοδο τα `tokens` και τα `labels` και εξετάζει αν για κάποιο token, το `word id` που του αντιστοιχεί (το ελέγχει μέσω της μεθόδου `word_ids()`) είναι `None`. Αν ισχύει αυτό τότε αποδίδει στο token αυτό το `label -100` το οποίο για την PyTorch σημαίνει ότι θα πρέπει να αγνοήσει αυτό το token στις περαιτέρω αναλύσεις. Καθώς ο BERT tokenizer μπορεί να σπάσει ένα token σε μικρότερα subtokens, θέλουμε μόνο το 1^ο subtoken να λαμβάνει κανονικό label και όλα τα subtokens του ίδιου token να παίρνουν `-100`. Αυτό γίνεται στο `elif` κομμάτι της μεθόδου. Κατά συνέπεια, τελικά η μέθοδος `EvaluateModel` θα αγνοήσει όλα τα tokens με `label -100` και επομένως θα εκτιμήσει το `accuracy` του μοντέλου μόνο με τα υπόλοιπα.

Ερώτημα 4

Προκειμένου να «παγώσουμε» τα βάρη του προεκπαιδευμένου μοντέλου BERT, πρέπει να προσθέσουμε τις ακόλουθες γραμμές κώδικα:

```
# By setting requires_grad to False we make sure that the parameters will not be changed (aka remain frozen) during the training process
for param in model.base_model.parameters():
    param.requires_grad = False
```

Ωστόσο, στο τελευταίο επίπεδο (classifier) θέλουμε να γίνουν αλλαγές επομένως θα προσαρμόσουμε κατάλληλα τον optimizer:

```
# Only the classifier (final) level parameters will be changed by the optimizer
optimizer = optim.AdamW(params=model.classifier.parameters(), lr=LR)
```

Με το ακόλουθο κομμάτι κώδικα βλέπουμε τελικά πόσα parameters έγιναν freeze και πόσα άλλαξαν:

```
frozen_params = 0
non_frozen_params = 0

for param in model.parameters():
    if param.requires_grad:
        non_frozen_params += param.numel()
    else:
        frozen_params += param.numel()

print("Number of frozen parameters: ", frozen_params)
print("Number of non-frozen parameters: ", non_frozen_params)
```

Το αποτέλεσμα:

```
Number of frozen parameters: 108891648
Number of non-frozen parameters: 6921
```

Η εκπαίδευση του μοντέλου με αυτή την τροποποίηση περιορίστηκε σημαντικά, ιδιαίτερα στο macro-accuracy, όπως φαίνεται από τα αποτελέσματα της εκπαίδευσης:

```
Test Accuracy : 0.848

Test Macro-Accuracy : 0.196
```

Ερώτημα 5

Για αυτό το ερώτημα ενώνουμε το training set και το validation set και εκπαιδεύουμε εκ νέου το μοντέλο. Οι αλλαγές που θα πρέπει να γίνουν είναι οι εξής:

Αρχικά προσθέτουμε τα validation και τα test sentences στο κομμάτι του κώδικα που καλείται η μέθοδος load_sentences:

```
train_sentences = load_sentences(base_path + 'train.txt') + load_sentences(base_path + 'valid.txt')
test_sentences = load_sentences(base_path + 'test.txt')
```

Το valid_dataset δεν χρειάζεται πλέον οπότε κάνουμε comment τη σχετική γραμμή στον κώδικα:

```
train_dataset = [encode(sentence) for sentence in train_sentences]
# valid_dataset = [encode(sentence) for sentence in valid_sentences]
test_dataset = [encode(sentence) for sentence in test_sentences]
```

Το ίδιο ισχύει και για τον valid_loader:

```
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
# valid_loader = torch.utils.data.DataLoader(valid_dataset, batch_size=BATCH_SIZE)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=BATCH_SIZE)
```

Τέλος, κατά το training του μοντέλου δεν υπάρχει πλέον validation στο τέλος κάθε εποχής, επομένως αφαιρούμε και αυτές τις γραμμές του κώδικα:

```
# train the model
print('training the model')
for epoch in tqdmn(range(EPOCHS)):
    model.train()
    print('epoch', epoch+1)
    # iterate through each batch of the train data
    for i, batch in enumerate(tqdmn(train_loader)):
        # move the batch tensors to the same device as the model
        batch = { k: v.to(device) for k, v in batch.items() }
        # send 'input_ids', 'attention_mask' and 'labels' to the model
        outputs = model(**batch)
        loss = outputs[0]
        # set the gradients to zero
        optimizer.zero_grad()
        # propagate the loss backwards
        loss.backward()
        # update the model weights
        optimizer.step()
    # calculate performance on validation set
    # Y_actual, Y_preds = EvaluateModel(model, valid_loader)
    # print("\nValidation Accuracy : {:.3f}".format(accuracy_score(Y_actual, Y_preds)))
    # print("\nValidation Macro-Accuracy : {:.3f}".format(balanced_accuracy_score(Y_actual, Y_preds)))
```

Με τις παραπάνω αλλαγές βλέπουμε ότι το accuracy και το macro-accuracy δεν διαφοροποιήθηκαν σημαντικά σε σχέση με το 1^ο ερώτημα:

```
Test Accuracy : 0.980
```

```
Test Macro-Accuracy : 0.907
```

Ερώτημα 6

Οι αλλαγές που θα πρέπει να κάνουμε έτσι ώστε ο κώδικας να πραγματοποιεί επισημείωση μέρους-του-λόγου (POS tagging) είναι οι εξής:

Εδώ θα πρέπει να αντικαταστήσουμε το `ner_tags` με `pos_tags`:

```
# build tagset and tag ids
tags = [sentence['pos_tags'] for sentence in train_sentences]
tagmap = build_vocab_from_iterator(tags)
tagset = set([item for sublist in tags for item in sublist])
print('Tagset size:', len(tagset))
```

Το ίδιο και στη μέθοδο `encode`:

```
def encode(sentence):
    encodings = tokenizer(sentence['tokens'], truncation=True, padding='max_length', is_split_into_words=True)
    labels = align_label(encodings, sentence['pos_tags'])
    return { 'input_ids': torch.LongTensor(encodings.input_ids), 'attention_mask': torch.LongTensor(encodings.attention_mask), 'labels': torch.LongTensor(labels) }
```

Ο υπόλοιπος κώδικας παραμένει ως έχει. Εδώ μία πρόταση που το μοντέλο δεν κατάφερε να προβλέψει σωστά τα `pos tags` ήταν η εξής:

Token	Actual Tag	Predicted Tag
SOCCER	NN	NN
-	:	:
JAPAN	NNP	NNP
GET	VB	NNP
LUCKY	NNP	NNP
WIN	NNP	NNP
,	,	,
CHINA	NNP	NNP
IN	IN	IN
SURPRISE	DT	NNP
DEFEAT	NN	NN
.	.	.

Όπως βλέπουμε στο token 'GET' το μοντέλο απέδωσε το tag 'NNP' ενώ στην πραγματικότητα θα έπρεπε να δώσει 'VB' και στο token 'SURPRISE' έδωσε 'NNP' ενώ θα έπρεπε να έχει δώσει 'DT'.

Στην πρόταση από το Wall Street Journal είχαμε τα εξής αποτελέσματα:

The actual and predicted tags for the tokens of this sentence were:

Token	Actual Tag	Predicted Tag
Zurich	NNP	NNP
and	CC	CC
some	DT	DT
other	JJ	JJ
insurers	NNS	NNS
with	IN	IN
big	JJ	JJ
fleets	NNS	NNS
of	IN	IN
engineers	NNS	NNS
are	VBP	VBP
advising	VBG	VBG
companies	NNS	NNS
on	IN	IN
how	WRB	WRB
to	TO	TO
fortify	VB	VB
their	PRP\$	PRP\$
properties	NNS	NNS
.	.	.

Εδώ φαίνεται ότι το model δεν έκανε κανένα λάθος και πρόβλεψε σωστά όλα τα tags.

Το accuracy και το macro-accuracy του μοντέλου στο POS tagging ήταν:

Test Accuracy : 0.941

Test Macro-Accuracy : 0.861

Ερώτημα 7

Οι αλλαγές που θα πρέπει να κάνουμε έτσι ώστε ο κώδικας να πραγματοποιεί ανάγνωση ορίων φράσεων (text chunking) είναι αντίστοιχες με αυτές του προηγούμενου ερωτήματος. Συγκεκριμένα:

```
# build tagset and tag ids
tags = [sentence['chunk_tags'] for sentence in train_sentences]
tagmap = build_vocab_from_iterator(tags)
tagset = set([item for sublist in tags for item in sublist])
print('tagset size: ',len(tagset))
```

και

```
def encode(sentence):
    encodings = tokenizer(sentence['tokens'], truncation=True, padding='max_length', is_split_into_words=True)
    labels = align_label(encodings, sentence['chunk_tags'])
    return { 'input_ids': torch.LongTensor(encodings.input_ids), 'attention_mask': torch.LongTensor(encodings.attention_mask), 'labels': torch.LongTensor(labels) }
```

Μία πρόταση στην οποία το μοντέλο δεν προέβλεψε σωστά τα tags ήταν η:

Token	Actual Tag	Predicted Tag
SOCCER	B-NP	B-NP
-	O	O
JAPAN	B-NP	B-NP
GET	B-VP	I-NP
LUCKY	B-NP	I-NP
WIN	I-NP	I-NP
,	O	O
CHINA	B-NP	B-NP
IN	B-PP	B-PP
SURPRISE	B-NP	B-NP
DEFEAT	I-NP	I-NP
.	O	O

Όπως βλέπουμε στο token 'GET' έβαλε I-NP, ενώ θα έπρεπε να έχει βάλει B-VP.

Στην πρόταση από το Wall Street Journal υπήρχαν επίσης ορισμένα λάθη:

Token	Actual Tag	Predicted Tag
Zurich	B-NP	B-NP
and	O	O
some	B-NP	B-NP
other	I-NP	I-NP
insurers	B-NP	I-NP
with	B-PP	B-PP
big	B-NP	B-NP
fleets	I-NP	I-NP
of	B-PP	B-PP
engineers	B-NP	B-NP
are	B-VP	B-VP
advising	B-VP	I-VP
companies	B-NP	B-NP
on	B-PP	B-PP
how	B-ADVP	B-ADVP
to	B-VP	B-VP
fortify	B-VP	I-VP
their	B-NP	B-NP
properties	I-NP	I-NP
.	O	O

Συγκεκριμένα, στο token 'insurers' δόθηκε το tag 'I-NP' αντί για το 'B-NP', στο 'advising' δόθηκε 'I-VP' αντί για 'B-VP' και στο 'fortify' δόθηκε 'B-VP' αντί για 'I-VP'.

Το accuracy και το macro-accuracy του μοντέλου στο text chunking ήταν:

Test Accuracy : 0.953

Test Macro-Accuracy : 0.580

Ερώτημα 8

Οι αλλαγές που πρέπει να γίνουν στον κώδικα έτσι ώστε να χρησιμοποιείται το μοντέλο Roberta-base είναι οι εξής:

Αρχικά πρέπει να κάνουμε import το μοντέλο και τον tokenizer από τη βιβλιοθήκη transformers:

```
# dependencies
import torch
import torch.optim as optim
from torchtext.vocab import build_vocab_from_iterator
from transformers import RobertaForTokenClassification, RobertaTokenizerFast
from sklearn.metrics import accuracy_score, balanced_accuracy_score, classification_report
import tqdm
tqdmn = tqdm.notebook.tqdm|
```

Στη συνέχεια πρέπει να φορτώσουμε τον tokenizer του Roberta:

```
# load the roberta tokenizer
roberta_version = 'roberta-base'
tokenizer = RobertaTokenizerFast.from_pretrained(roberta_version, add_prefix_space=True)
```

Και τέλος να φορτώσουμε το μοντέλο:

```
# initialize the model including a classification layer with num_labels classes
print('initializing the model')
model = RobertaForTokenClassification.from_pretrained(roberta_version, num_labels=len(tagset))
model.to(device)
optimizer = optim.AdamW(params=model.parameters(), lr=LR)
```

Το μοντέλο αυτό φαίνεται ότι τελικά έχει μεγαλύτερο accuracy και macro-accuracy σε σχέση με το αρχικό μας μοντέλο με το BERT:

```
Test Accuracy : 0.984
```

```
Test Macro-Accuracy : 0.926
```

Ερώτημα 9

Με τα αποτελέσματα που λάβαμε από τα προηγούμενα ερωτήματα για τις δύο προτάσεις (την τυχαία πρόταση που είχε λάθος tags και αυτή που επιλέξαμε εμείς από το Wall Street Journal) θα εξετάσουμε ποια είναι η απάντηση του ChatGPT για αυτές τις προτάσεις και θα συγκρίνουμε τα αποτελέσματα.

Ξεκινώντας με την πρόταση «SOCCER - JAPAN GET LUCKY WIN, CHINA IN SURPRISE DEFEAT.» με zero-shot prompting έχουμε:

Prompt:

SOCCER - JAPAN GET LUCKY WIN, CHINA IN SURPRISE DEFEAT.
Perform Named Entity Recognition Tagging, POS tagging and text chunking on the tokens of the above sentence.

Απάντηση:

Sure! Let's perform Named Entity Recognition (NER) tagging, Part-of-Speech (POS) tagging, and text chunking on the tokens in the given sentence.

Tokens:

1. SOCCER
2. -
3. JAPAN
4. GET
5. LUCKY
6. WIN
7. ,
8. CHINA
9. IN
10. SURPRISE
11. DEFEAT
12. .

NER Tagging:

1. SOCCER -> O (Not a named entity)
2. - -> O (Not a named entity)
3. JAPAN -> B-LOC (Named entity - Location)
4. GET -> O (Not a named entity)
5. LUCKY -> O (Not a named entity)
6. WIN -> O (Not a named entity)
7. , -> O (Not a named entity)
8. CHINA -> B-LOC (Named entity - Location)
9. IN -> O (Not a named entity)
10. SURPRISE -> O (Not a named entity)

11. DEFEAT -> O (Not a named entity)
12. . -> O (Not a named entity)

POS Tagging:

1. SOCCER -> NN (Noun, singular or mass)
2. - -> HYPH (Punctuation mark, hyphen)
3. JAPAN -> NNP (Proper noun, singular)
4. GET -> VB (Verb, base form)
5. LUCKY -> JJ (Adjective)
6. WIN -> NN (Noun, singular or mass)
7. , -> , (Punctuation mark, comma)
8. CHINA -> NNP (Proper noun, singular)
9. IN -> IN (Preposition or subordinating conjunction)
10. SURPRISE -> NN (Noun, singular or mass)
11. DEFEAT -> NN (Noun, singular or mass)
12. . -> . (Punctuation mark, sentence closer)

Text Chunking:

1. SOCCER -> Noun Phrase
2. - -> Not a chunk
3. JAPAN -> Noun Phrase
4. GET LUCKY WIN -> Verb Phrase
5. , -> Not a chunk
6. CHINA -> Noun Phrase
7. IN -> Not a chunk
8. SURPRISE DEFEAT -> Noun Phrase
9. . -> Not a chunk

Please note that the chunking is subjective and may vary based on the specific chunking rules used.

Σχολιασμός των αποτελεσμάτων:

Για το NER tagging βλέπουμε ότι όπως και το δικό μας μοντέλο, το ChatGPT θεωρεί ότι στο token “CHINA” αντιστοιχεί το B-LOC, κάτι που είναι μάλλον σωστό καθώς η Κίνα είναι περιοχή. Άρα εδώ θα μπορούσαμε να πούμε ότι τόσο το μοντέλο μας όσο και το ChatGPT έχουν δίκιο και πιθανώς έχει λάθος το αρχικό tagging.

Για το POS tagging το ChatGPT βρήκε σωστά το tag για το token “GET”, ενώ το δικό μας μοντέλο είχε κάνει λάθος. Και τα δύο μοντέλα έκαναν λάθος για το “SURPRISE”, καθώς κανένα από τα δύο δεν έδωσε το σωστό tag “DT”.

Τέλος, στο text chunking η απάντηση του ChatGPT δεν ήταν αυτό που θα περιμέναμε και γι’ αυτό πιθανώς ενοχοποιείται το zero-shot prompting. Ωστόσο, το μοντέλο αναφέρει ότι το chunking είναι υποκειμενικό και μπορεί να διαφέρει ανάλογα με τους κανόνες που χρησιμοποιούμε κάθε φορά.

Για την πρόταση του Wall Street Journal επαναλαμβάνουμε την ίδια διαδικασία και λαμβάνουμε την εξής απάντηση από το ChatGPT:

Απάντηση:

Sure! Let's perform Named Entity Recognition (NER) tagging, Part-of-Speech (POS) tagging, and text chunking on the tokens in the given sentence.

Tokens:

1. Zurich
2. and
3. some
4. other
5. insurers
6. with
7. big
8. fleets
9. of
10. engineers
11. are
12. advising
13. companies
14. on
15. how
16. to
17. fortify
18. their
19. properties
20. .

NER Tagging:

1. Zurich -> B-ORG (Named entity - Organization)
2. and -> O (Not a named entity)
3. some -> O (Not a named entity)
4. other -> O (Not a named entity)
5. insurers -> O (Not a named entity)
6. with -> O (Not a named entity)
7. big -> O (Not a named entity)
8. fleets -> O (Not a named entity)
9. of -> O (Not a named entity)
10. engineers -> O (Not a named entity)
11. are -> O (Not a named entity)
12. advising -> O (Not a named entity)
13. companies -> O (Not a named entity)
14. on -> O (Not a named entity)
15. how -> O (Not a named entity)

16. to -> O (Not a named entity)
17. fortify -> O (Not a named entity)
18. their -> O (Not a named entity)
19. properties -> O (Not a named entity)
20. . -> O (Not a named entity)

POS Tagging:

1. Zurich -> NNP (Proper noun, singular)
2. and -> CC (Coordinating conjunction)
3. some -> DT (Determiner)
4. other -> JJ (Adjective)
5. insurers -> NNS (Noun, plural)
6. with -> IN (Preposition or subordinating conjunction)
7. big -> JJ (Adjective)
8. fleets -> NNS (Noun, plural)
9. of -> IN (Preposition or subordinating conjunction)
10. engineers -> NNS (Noun, plural)
11. are -> VBP (Verb, non-3rd person singular present)
12. advising -> VBG (Verb, gerund or present participle)
13. companies -> NNS (Noun, plural)
14. on -> IN (Preposition or subordinating conjunction)
15. how -> WRB (Wh-adverb)
16. to -> TO (To)
17. fortify -> VB (Verb, base form)
18. their -> PRP\$ (Possessive pronoun)
19. properties -> NNS (Noun, plural)
20. . -> . (Punctuation mark, sentence closer)

Text Chunking:

1. Zurich -> Noun Phrase
2. and -> Not a chunk
3. some -> Not a chunk
4. other -> Not a chunk
5. insurers -> Noun Phrase
6. with big fleets of engineers -> Prepositional Phrase
7. are advising -> Verb Phrase
8. companies -> Noun Phrase
9. on -> Not a chunk
10. how to fortify their properties -> Noun Phrase
11. . -> Not a chunk

Again, please note that the chunking is subjective and may vary based on the specific chunking rules used.

Σχολιασμός των αποτελεσμάτων:

Εδώ βλέπουμε ότι η απάντηση του ChatGPT συμφωνεί με το μοντέλο μας για το NER tagging καθώς και τα δύο μοντέλα έχουν δώσει σωστά το tag “B-ORG” στο token “Zurich”.

Στο POS Tagging τόσο το ChatGPT όσο και το μοντέλο μας προέβλεψαν τέλεια όλα τα tags.

Στο text chunking και εδώ η απάντηση του ChatGPT δεν είναι αυτή που θα περιμέναμε και πιθανώς ευθύνεται και πάλι το zero-shot prompting.

Ερώτημα 10

Σε αυτό το ερώτημα δοκιμάζουμε το few-shot prompting για να δούμε αν βελτιώνονται οι απαντήσεις του ChatGPT σε σχέση με το zero-shot prompting. Ξεκινάμε αρχικά από την πρόταση «SOCCER - JAPAN GET LUCKY WIN, CHINA IN SURPRISE DEFEAT.» που είχε λάθος tags στο μοντέλο μας για όλα τα taggings.

Στο few-shot prompting θα δώσουμε στο ChatGPT τις πρώτες 5 προτάσεις από τα train_sentences μαζί με τα tags του και θα ζητήσουμε να χρησιμοποιήσει την ίδια μέθοδο για να μας δώσει τα tags στην πρόταση που ζητάμε.

Prompt:

```
[{'tokens': ['EU',
'rejects',
'German',
'call',
'to',
'boycott',
'British',
'lamb',
'.'],
'pos_tags': ['NNP', 'VBZ', 'JJ', 'NN', 'TO', 'VB', 'JJ', 'NN', '.'],
'chunk_tags': ['B-NP',
'B-VP',
'B-NP',
'I-NP',
'B-VP',
'I-VP',
'B-NP',
'I-NP',
'O'],
'ner_tags': ['B-ORG', 'O', 'B-MISC', 'O', 'O', 'O', 'B-MISC', 'O', 'O']},
{'tokens': ['Peter', 'Blackburn'],
'pos_tags': ['NNP', 'NNP'],
'chunk_tags': ['B-NP', 'I-NP'],
'ner_tags': ['B-PER', 'I-PER']},
{'tokens': ['BRUSSELS', '1996-08-22'],
'pos_tags': ['NNP', 'CD'],
'chunk_tags': ['B-NP', 'I-NP'],
'ner_tags': ['B-LOC', 'O']},
{'tokens': ['The',
'European',
'Commission',
'said',
'on',
```

'Thursday',
'it',
'disagreed',
'with',
'German',
'advice',
'to',
'consumers',
'to',
'shun',
'British',
'lamb',
'until',
'scientists',
'determine',
'whether',
'mad',
'cow',
'disease',
'can',
'be',
'transmitted',
'to',
'sheep',
''],
'pos_tags': ['DT',
'NNP',
'NNP',
'VBD',
'IN',
'NNP',
'PRP',
'VBD',
'IN',
'JJ',
'NN',
'TO',
'NNS',
'TO',
'VB',
'JJ',
'NN',
'IN',
'NNS',
'VBP',
'IN',

'JJ',
'NN',
'NN',
'MD',
'VB',
'VBN',
'TO',
'NN',
'.'],
'chunk_tags': ['B-NP',
'I-NP',
'I-NP',
'B-VP',
'B-PP',
'B-NP',
'B-NP',
'B-VP',
'B-PP',
'B-NP',
'I-NP',
'B-PP',
'B-NP',
'B-VP',
'I-VP',
'B-NP',
'I-NP',
'B-SBAR',
'B-NP',
'B-VP',
'B-SBAR',
'B-NP',
'I-NP',
'I-NP',
'B-VP',
'I-VP',
'I-VP',
'B-PP',
'B-NP',
'O'],
'ner_tags': ['O',
'B-ORG',
'I-ORG',
'O',
'O',
'O',
'O']

'O',
'O',
'B-MISC',
'O',
'O',
'O',
'O',
'O',
'B-MISC',
'O',
'O',
'O',
'O',
'O',
'O',
'O',
'O',
'O',
'O',
'O',
'O',
'O'],
{'tokens': ['Germany',
"s",
'representative',
'to',
'the',
'European',
'Union',
"s",
'veterinary',
'committee',
'Werner',
'Zwingmann',
'said',
'on',
'Wednesday',
'consumers',
'should',
'buy',
'sheepmeat',
'from',
'countries',
'other',
'than',

'Britain',
'until',
'the',
'scientific',
'advice',
'was',
'clearer',
'.'],
'pos_tags': ['NNP',
'POS',
'NN',
'TO',
'DT',
'NNP',
'NNP',
'POS',
'JJ',
'NN',
'NNP',
'NNP',
'VBD',
'IN',
'NNP',
'NNS',
'MD',
'VB',
'NN',
'IN',
'NNS',
'JJ',
'IN',
'NNP',
'IN',
'DT',
'JJ',
'NN',
'VBD',
'JJR',
'.'],
'chunk_tags': ['B-NP',
'B-NP',
'I-NP',
'B-PP',
'B-NP',
'I-NP',
'I-NP',

'B-NP',
'I-NP',
'I-NP',
'I-NP',
'I-NP',
'B-VP',
'B-PP',
'B-NP',
'I-NP',
'B-VP',
'I-VP',
'B-NP',
'B-PP',
'B-NP',
'B-ADJP',
'B-PP',
'B-NP',
'B-SBAR',
'B-NP',
'I-NP',
'I-NP',
'B-VP',
'B-ADJP',
'O'],
'ner_tags': ['B-LOC',
'O',
'O',
'O',
'O',
'B-ORG',
'I-ORG',
'O',
'O',
'O',
'B-PER',
'I-PER',
'O',
'O',
'O',
'O',
'O',
'O',
'O',
'O',
'O']

'O',
'B-LOC',
'O',
'O',
'O',
'O',
'O',
'O',
'O']}]

Using the provided data for the above 5 sentences as a guide, give me the pos_tags, ner_tags and chunk_tags for the sentence:

SOCCER - JAPAN GET LUCKY WIN, CHINA IN SURPRISE DEFEAT.

Απάντηση:

Based on the provided data, here are the POS tags, NER tags, and chunk tags for the sentence "SOCCER - JAPAN GET LUCKY WIN, CHINA IN SURPRISE DEFEAT":

POS tags: ['NN', 'HYPH', 'NNP', 'NNP', 'VB', 'JJ', 'NN', 'NNP', 'IN', 'NNP', 'NNP', '.']

NER tags: ['O', 'O', 'B-LOC', 'O', 'O', 'O', 'O', 'B-LOC', 'O', 'B-LOC', 'O', 'O']

Chunk tags: ['B-NP', 'I-NP', 'B-NP', 'I-NP', 'B-VP', 'B-ADJP', 'I-ADJP', 'B-NP', 'B-PP', 'B-NP', 'I-NP', 'O']

Please note that the chunk tags may vary depending on the specific chunking scheme used.

Σχολιασμός των αποτελεσμάτων:

Για το NER tagging βλέπουμε ότι εδώ το ChatGPT έχει κάνει λάθος καθώς πρόσθεσε B-LOC στο token "SURPRISE" κάτι που δεν είχε κάνει με το zero-shot prompting. Τα υπόλοιπα tags είναι τα ίδια με το zero shot prompting. Το δικό μας μοντέλο είχε προβλέψει σωστά όλα τα tags εκτός από το "CHINA" στο οποίο είχε δώσει tag "B-LOC" αντί για "B-PER".

Στο POS tagging η απάντηση του ChatGPT έχει επίσης περισσότερα λάθη αφού σε πολλαπλά tokens δεν έχει δώσει το σωστό tag. Άρα μέχρι τώρα θα λέγαμε ότι το few-shot prompting έκανε το μοντέλο του ChatGPT να μας δώσει χειρότερη απάντηση σε σχέση με το zero-shot prompting.

Τέλος, στο text chunking υπάρχει σαφής βελτίωση σε σχέση με το zero-shot prompting καθώς εκτός ελαχίστων εξαιρέσεων το ChatGPT κατάφερε να μας δώσει σωστά αποτελέσματα, τα οποία μάλιστα είναι πολύ κοντά σε αυτά που έδωσε το δικό μας μοντέλο.

Στη συνέχεια συνεχίζουμε με την πρόταση από το Wall Street Journal. Χρησιμοποιώντας το ίδιο prompt όπως και πριν (με αλλαγή μόνο την πρόταση του WSJ) παίρνουμε την εξής απάντηση από το ChatGPT.

Απάντηση:

Based on the provided data, here are the POS tags, NER tags, and chunk tags for the sentence "Zurich and some other insurers with big fleets of engineers are advising companies on how to fortify their properties":

POS tags: ['NNP', 'CC', 'DT', 'JJ', 'NNS', 'IN', 'JJ', 'NNS', 'VBP', 'VBG', 'NNS', 'IN', 'WRB', 'TO', 'VB', 'PRP\$', 'NNS', '.']
NER tags: ['B-ORG', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
Chunk tags: ['B-NP', 'O', 'B-NP', 'I-NP', 'I-NP', 'B-PP', 'B-NP', 'I-NP', 'B-VP', 'I-VP', 'I-VP', 'B-PP', 'B-ADVP', 'B-TO', 'B-VP', 'B-NP', 'I-NP', 'O']

Please note that the NER tags indicate no named entities in this sentence, and the chunk tags may vary depending on the specific chunking scheme used.

Σχολιασμός των αποτελεσμάτων:

Εδώ βλέπουμε ότι η απάντηση του ChatGPT είναι και πάλι σωστή και συμφωνεί με το μοντέλο μας για το NER tagging, καθώς και τα δύο μοντέλα έχουν δώσει σωστά το tag “B-ORG” στο token “Zurich”.

Στο POS Tagging το few-shot prompting οδήγησε σε ορισμένα λάθη, κάτι που δεν είχε συμβεί στο zero-shot prompting. Το μοντέλο μας είχε προβλέψει σωστά όλα τα tags εδώ.

Στο text chunking υπάρχει σαφής βελτίωση σε σχέση με το zero-shot prompting καθώς το chat απέτυχε να προβλέψει σωστά ελάχιστα tokens.