令和 6 年度茨城大学大学院理工学研究科情報工学専攻

修士学位論文

Zero Shot Cross Lingual Transfer の評判分析における多言語モデルと MT + 単言語モデルの性能比較

所属　情報工学専攻

著者　佐藤匠真 (23NM724Y)

指導教員　新納浩幸教授

令和 7 年 2 月 3 日 (火)

令和 6 年度茨城大学大学院理工学研究科情報工学専攻　修士学位論文

Zero Shot Cross Lingual Transfer の評判分析における多言語モデルと MT + 単言語モデルの性能比較

著者

佐藤匠真 (23NM724Y)

指導教員

新納浩幸教授

論文要旨

多言語モデルを利用した Zero shot Cross Lingual Transfer とは言語 A の訓練データを利用して，多言語モデルを fine-tuning し，その fine-tuning したモデルを利用して言語 B のテストデータを解析する技術である．一方，言語 A から言語 B の翻訳機と言語 B の単言語モデルがあれば，言語 A の訓練データを言語 B に翻訳し，その翻訳したデータから言語 B の単言語モデルを fine-tuning できる．そしてその fine-tuning したモデルを利用して，言語 B のテストデータを解析できる．このように翻訳機が利用できれば単言語モデルであっても，Zero shot Cross Lingual transfer と同等の処理が可能になる．多言語モデルの利用と MT+ 単言語モデルの利用のどちらが有効であるかは明らかではない．本稿では言語 A を英語，言語 B を日本語，またタスクを評判分析として，上記 2 つの手法（多言語モデル vs MT + 単言語モデル）の性能比較を行い，先の未解明事項について考察する．

しかし，近年の大規模言語モデルは急速に発展しており，GPT-3.5 や GPT-4 などの大規模なトランスフォーマーモデルは，従来の教師あり学習の枠を超え，事前学習された知識を活用することで，特定のタスクに対してデータセットを準備せずとも高い精度を達成できる Zero Shot 学習が可能となっている．本稿では，Zero Shot Cross Lingual Transfer と LLM を利用した Zero Shot の評判分析における性能について比較する．

実験では，タスクを Amazon レビューの評判分析として比較した．実験の結果，MT+ 単言語モデルの性能が最も高かった．また，LLM を利用した Zero Shot 評判分析の方は Zero Shot Cross Lingual Transfer や教師あり学習と同等以上の精度であることが確認できた．LLM を利用した日本語の Zero Shot 評判分析においては Chain of Thought による精度の向上は見られなかった．

Master's Thesis in Scholastic 2024, Major in Computer and Information Sciences,
Granduate School of Science and Engineering, Ibaraki University

Comparison of the Performance of Multilingual Models and MT + Monolingual Models in Zero Shot Cross Lingual Transfer for Sentiment Analysis

Author : Takuma Sato (23NM724Y)

Adviser : Prof. Hiroyuki Shinnou

Abstract

Zero-shot Cross-Lingual Transfer using multilingual models involves fine-tuning a multilingual model on training data from language A and using it to analyze test data in language B. On the other hand, with a machine translation (MT) system from language A to language B and a monolingual model in language B, training data in language A can be translated into language B, the monolingual model fine-tuned using the translated data, and test data in language B analyzed using the fine-tuned model. This suggests that even monolingual models can achieve results comparable to Zero-shot Cross-Lingual Transfer when paired with an MT system. However, it remains unclear which approach—multilingual models or MT combined with monolingual models—is more effective. In this study, we investigate these two methods (multilingual models vs. MT + monolingual models) for sentiment analysis, using English as language A and Japanese as language B, and evaluate their performance. Recent advancements in large-scale language models (LLMs), such as GPT-3.5 and GPT-4, have transcended traditional supervised learning frameworks, achieving high accuracy in specific tasks through Zero-shot learning without task-specific datasets. This study also compares the performance of Zero-shot Cross-Lingual Transfer with that of sentiment analysis conducted using LLMs in a Zero-shot setting. Experiments on sentiment analysis of Amazon reviews showed that the MT + monolingual model performed best, while Zero-shot sentiment analysis using LLMs achieved performance comparable to both Zero-shot Cross-Lingual Transfer and supervised learning. In addition, in Japanese Zero-shot sentiment analysis using LLMs, no accuracy improvement was observed with the Chain of Thought approach.

目次

表目次

図目次

第 1 章

序論

多言語間の汎用的な特徴表現を学習する多言語の事前学習済みモデルとして，mBERT [1]，XLM [2]，XLM-RoBERTa-XL [3] などがこれまでに提案されてきた．このようなモデルの応用の一つとして Zero Shot Cross Lingual Transfer [4] がある．Zero Shot Cross Lingual Transfer は，あるタスクに対して言語 A で fine-tunining したモデルを別の言語 B の同じタスクに対してそのまま利用する手法である．通常，言語 A を英語などのメジャーな言語，言語 B はリソースが少ない，あるいはないようなマイナーな言語に設定することで，低資源言語の学習の問題解決に利用される．

また，言語 A から言語 B の翻訳機と言語 B の単言語モデルがあれば，言語 A の訓練データを言語 B に翻訳し，その翻訳したデータから言語 B の単言語モデルを fine-tuning できる．そしてその fine-tuning したモデルを利用して，言語 B のテストデータを解析できる．つまり翻訳機が利用出来れば，Zero Shot Cross Lingual Transfer と同等の処理が行える．本稿では翻訳機 MT が利用できる場合に多言語モデルを利用する方がよいのか，MT+ 単言語モデルを利用する方がよいのかを評判分析のタスクを通して調査する．

実験ではタスクを Amazon レビューの評判分析とし，上記言語 A を英語に，言語 B を日本語に設定した．多言語モデルは xlm-RoBERTa，日本語の単言語モデルは日本語 RoBERTa，英語の単言語モデルに RoBERTa を利用した．このような設定下で，以下の 3 つの手法を比較する．

- (手法 1)xlm-RoBERTa を訓練データである英文書で fine-tuning し，テストデータである和文書でテストする (Zero Shot Cross Lingual Transfer)
- (手法 2) 訓練データである英文書を翻訳機で和訳し，その和訳文書で日本語

RoBERTa を fine-tuning し，テストデータである和文書でテストする (MT ＋単言語)

- (手法 3) 訓練データである英文書を翻訳機で和訳し，その和訳文書ともとの英文書を合わせたデータで xlm-RoBERTa を fine-tuning し，テストデータである和文書でテストする (MT+ 多言語)

実験の結果，手法 3 より手法 1 の精度が高く，手法 2 の精度が最も高かった．

また, LLM を利用した Zero Shot 評判分析と性能を比較するために, GPT3.5-turbo [5] と GPT-4o と Swallow [6] で Amazon レビューの評判分析を行った．レビュー文書のラベルを 2 値分類および 4 値分類で評価した．また，極性判定以外の文書分類タスクについても性能調査を行った．具体的には，Livedoor ニュース記事の分類をタスクとして実験を行い，評判分析で使用したモデルと同じ LLM を利用した．実験の結果，極性判定の 2 値分類および 4 値分類において，Zero Shot 評判分析が RoBERTa の教師あり学習と同等以上の性能を示すことが確認された．一方で，文書分類タスクでは，Zero Shot 学習の精度が教師あり学習を下回る結果となった．また，LLM を利用した Zero Shot で日本語の Amazon レビューの評判分析を実施する際，LLM にその判定根拠を提示させると精度が低下することが確認された．

第 2 章

関連研究

2.1 ニューラルネットワーク

ニューラルネットワークは，なんらかの変換を行う層の組み合わせによって構成される．各層は一般に，入力値に対して適当な線形変換と，活性化関数の合成変換が施された出力値を返す．活性化関数は，Sigmoid 関数や，ReLU 関数など様々な種類がある．ニューラルネットワークは，複数の層を組み合わせることによって高い表現能力を示す．

各層で行われる変換は，その層が持つパラメータに依存する．これらのパラメータが学習の過程で調整されることで，ニューラルネットワークは適切に問題を解くことができるようになる．パラメータを調整する際に，必要となる現在のパラメータの「良さ」を表す基準が必要である．この「良さ」は

- そのパラメータを持つニューラルネットワークが表現する入出力関係
- タスクで獲得したい理想的な入出力関係

を比較することで確認でき，ズレが少なければ少ないほど良いパラメータである．このズレを定量的に表現する関数を損失関数という．

2.1.1 分類問題

自然言語処理の多くのタスクは，与えられたデータに対して，与えられた複数のカテゴリーの中から，そのデータに対応するカテゴリーを決める分類問題として扱うことができる．

カテゴリーの数を N, データを表現するベクトルを x とし，カテゴリーを 1 から N の整数で表現したものをラベルといい，l とする．分類問題とはデータ x が与えられたときに，そのラベル l を予測するモデルを作ることである．

分類問題はニューラルネットワークを用いて解くことができる．ニューラルネットワークの出力を

$$y = F(x, \theta)$$

と置く．θ は調整可能なパラメータである．出力 y はカテゴリーの数と同じ次元である．出力 y の各要素の値は，それぞれカテゴリーへの予測の確度を表すものである．それを分類スコアと呼ぶ．y の j 番目の要素の値は j 番目のカテゴリーに対する分類スコアを表す．この分類スコアが最も高いカテゴリーをニューラルネットワークの予測とする．図 2.1 を例にするとラベル 2 の分類スコアが最も高いため，ニューラルネットワークはラベル l が 2 と予測する．

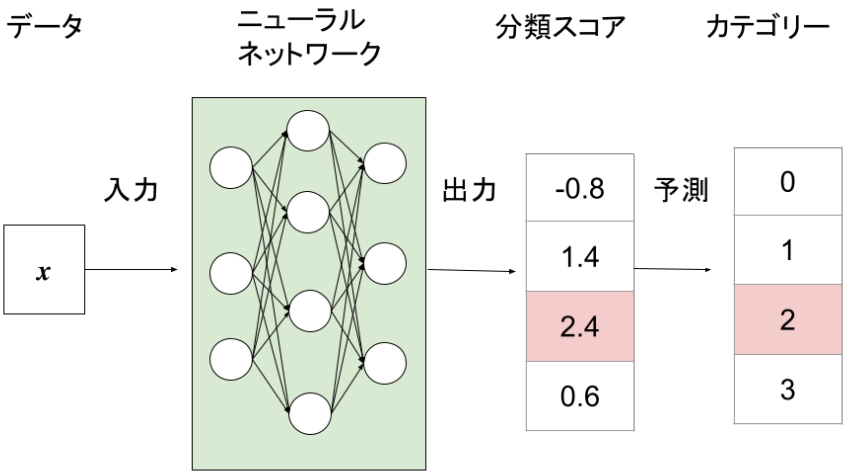

図 2.1: 分類問題の例.

2.2　Zero Shot Learning

Zero Shot Learning は機械が見たことないもの，知らないものを予測するための機械学習の技術の一つである．通常の分類タスクの場合，例えば，「縞模様」と「馬」の画像を学習して未学習の「シマウマ」が画像が入力された場合，縞模様か馬のクラスに分類される．しかし，Zero Shot Learning の場合，未知クラスのデータ「シマウマ」が入力さ

れた場合でも未知の正しい「シマウマ」クラスに分類できる．これは，「シマウマ」の画像から「馬の特徴」と「縞柄の特徴」をエンコーダが獲得し，Word2Vec などの事前学習済みモデルで「シマウマ」が縞柄の馬であるという事前知識を取り込むことができているからである．

Zero Shot Learning の評価には主に

- 未知クラスに対する認識性能を測る Zero Shot Learning
- 既知クラスと未知クラスに対する認識性能を測る Generalized Zero Shot Learning

がある．

2.3　BERT

Bidirectional Encoder Representations from Transformers(BERT) [1] は 2018 年に Google より公開された事前学習済みモデルである．様々な言語タスクで当時の最先端のモデルの性能を大きく上回る性能を示した．BERT は文脈を考慮した分散表現を生成するモデルである．BERT 以前から文脈を考慮するモデルは存在していたが，BERT では Attention という手法で離れた位置の情報も取り入れることができる．BERT は Transformer というモデル [7] で提案された Transformer Encoder と呼ばれる Attention を用いたニューラルネットワークを用いている．Transformer Encoder のそれぞれの層はおもに，Multi-Head Attention と Feed Forward Network から構成される．

BERT の内部構造は図 2.2 のように，入力データがベクトル化され，層に渡されその出力が次の層に渡される．そして，最後の層の出力が BERT としての出力となる．

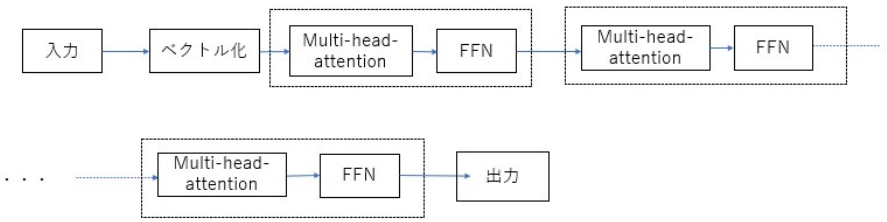

図 2.2: BERT の内部構造．

また，BERT には事前学習とファインチューニングの二つの学習の過程がある．

2.3.1　Scaled Dot-Product Attention

Scaled Dot-Product Attention は Multi-Head Attention の構成要素である．

例として，n 個のトークンで構成される文章を処理するとする．一つ目の層での i 番目のトークンに対応する出力をベクトル x_i で与えられるとする．それぞれの出力に対して行列 W^q,W^k,W^v で線形変換を行うことにより，クエリ q_i, キー k_i, バリュー v_i と呼ばれる三つの d 次元ベクトルを準備する．

$$q_i = x_i W^q$$
$$k_i = x_i W^k$$
$$v_i = x_i W^v$$

Scaled Dot-Product Attention ではそれぞれのトークンはこれらの三つのベクトルにより特徴づけられる．そして，それぞれのトークンに対してベクトル a_i を出力する．a_i はバリュー v_i の重み付き平均

$$a_i = \sum_{j=1}^{n} a_{n,j} v_j$$

で与えられるとする．重み $a_{i,j}$ は i 番目のトークンを処理する際に，j 番目のトークンの情報を重視する度合いを表している．重みはキーとクエリから決まる．i 番目のトークンを処理する時には，そのクエリと関連度が大きいようなキーを持つトークンほど出力に大きく寄与するようになる．Transformer では，Scaled Dot-Product でクエリとキーを評価する．

Scaled Dot-Product は q_i と k_j の内積を $\sqrt{d}$ で割って得られる $\hat{a}_{i,j}$ をスコアとして用いる．

得られたスコア $\hat{a}_{i.1}$,$\hat{a}_{i.2}$,...,$\hat{a}_{i.n}$ に Softmax 関数を適用することで、最終的に重み $a_{i.1}$,$a_{i.2}$,...,$a_{i.n}$ を得る．

$$[a_{i.1}, a_{i.2}, ..., a_{i.n}] = Softmax(\hat{a}_{i.1}, \hat{a}_{i.2}, ..., \hat{a}_{i.n})$$

また，異なるトークンに対する出力は別々に計算するのではなく，一つの行列演算で効率よく計算できる．入力、クエリ，キー，バリュー，出力のそれぞれを縦に結合した行列を，それぞれ X, Q, K, V, A と置く．出力 A は

$$A = Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d}})V$$

と表せる．Scaled Dot-Product Attention の処理の流れは図 2.3 のように表せる．

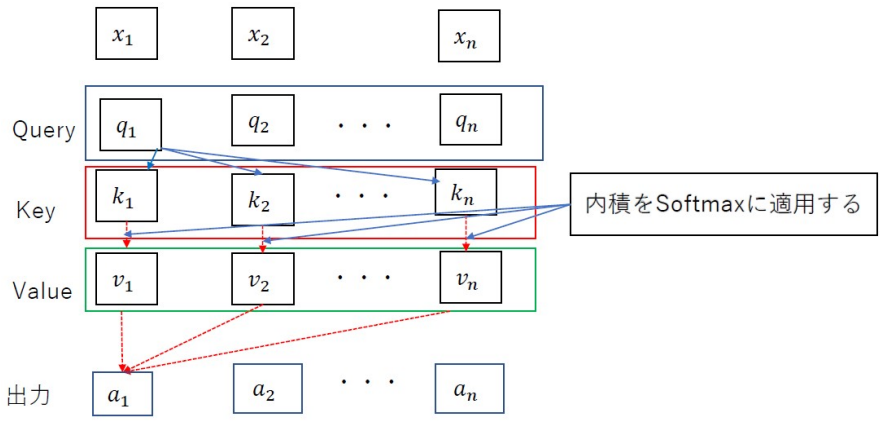

図 2.3: Scaled Dot-Product Attention の処理の流れ．

Attention ではあるトークンを処理するときに，すべてのトークンの情報を直接用いて出力を計算する．そのため，遠く離れたトークンの情報も適切に考慮することが可能である．

2.3.2　Multi-Head Attention

Multi-Head Attention は, クエリ, キー, バリューの組を複数用意し, それぞれの組に対して Scaled Dot-Produc Attention を適用し，最後に出力を一つに集約する手法である．Scaled Dot-Product Attention では，前の層の出力 x_i に対して，行列 W^Q, W^K, W^V を適用して q_i，k_i，v_i を得た．そのため，行列の組 $(W^Q_{(l)}$，$W^K_{(l)}$，$W^V_{(l)})(l = 1, 2, ..., h)$ を複数用意することで，クエリ，キー，バリューの組を複数作成することができる．それぞれの行列の組を用いて，Scaled Dot-Product Attention の出力 $a_i^{(l)}$ を得る．そして，それらを横に連結して一つのベクトルにし，さらに W_o により線形変換を行うことにより，最終的な出力 a_i を得る ($W^Q_{(l)}$，$W^K_{(l)}$，$W^V_{(l)}$，W^o はパラメータである)．

Multi-Head Attention もすべてのトークンをまとめて処理することができる．入力、クエリ，キー，バリューのそれぞれを縦に結合した行列を，それぞれ X, Q, K, V と置き，Scaled Dot-Product Attention の出力をすべてのトークンで縦に結合して行列にしたものを $A^{(l)}$ とし，Multi-Head Attention の出力を同様に行列にしたものを A と置く．すると，Multi-Head Attention の出力 A は

$$A = hstack(A^{(1)}, A^{(2)}, ..., A^{(h)})W^o$$

と表せる ($hstack$ は行列を横に結合する関数)．また Scaled Dot-Product Attention の出力 $A^{(l)}$ は

$$A^{(l)} = Attention(XW^Q_{(l)}, XW^K_{(l)}, XW^V_{(l)})$$

と表せる．$XW^Q_{(l)}$ は X がどの部分を処理するかを決め，$XW^K_{(l)}$ は X への注目の仕方を決め，$XW^V_{(l)}$ は X を回して出力の様子を調整する役割を持つ．

2.3.3　BERT の入出力

BERT では 1 つの文または 2 つの文章のペアが入力される．単一の文章の場合は文章をトークン化したのち，トークン列の先頭に [CLS] トークンを，末尾に [SEP] トークンを加える．2 つの文章のペアの場合は，一つ目の文章のトークン列と 2 つ目の文章のトークン列を並べ，文の境界に [SEP] トークンを置き，トークン列の先頭に [CLS] トークンを，末尾に [SEP] トークンを加える．

文章をトークン列に変換したあとに，トークンをベクトルに置き換えて BERT に入力する．そのため，以下のようにトークン，文章タイプ，文章中の位置それぞれに応じたベクトルの和を BERT へ入力する．

- サイズが m(語彙数) の行列 E^T を用意し，語彙中の j 番目のトークンが現れた場合，E^T の j 行目の行ベクトルに置き換える．このように文章中の i 番目のトークンをベクトルに置き換えたものを e^T_i と置く．
- 　サイズが (2,m) の行列 E^S を用意する．BERT に入力する文章が 1 文のみの場合はそれぞれのトークンを E^S の 1 行目の行ベクトルに置き換える．BERT に入力される文章が 2 文から構成される場合は，最初の文に含まれるトークンを E^S の 1 行目の行ベクトルに置き換え，2 つ目の文に含まれるトークンを E^S の 2 行目の行ベクトルに置き換える．このように文章中の i 番目のトークンをベクトルに置き換えたものを e^S_i と置く．
- BERT に入力可能な最大のトークンの数を L とし，サイズが (L,m) の行列 E^P を用意する．文章中の i 番目のトークンを行列 E^P の i 行目の行ベクトルに置き換え，e^P_i と置く．

最終的に，文章中の i 番目のトークンは 3 つの過程で得た 3 つのベクトルを足した e_i

に置き換えて BERT に入力する．

$$e_i = e_i^T + e_i^S + e_i^P$$

2.3.4　事前学習

BERT の学習は先述したように，事前学習と fine-tuning を行う．

事前学習には比較的容易に大量のデータを収集できるため，ラベルのついていないデータのみで行う．また，事前学習には Masked Language Model(MLM) と Next Sentence Prediction(NSP) の 2 つがある．

Masked Language Model は，ある単語を m 割の単語から予測するというタスクである．まず，入力されたトークンの 15% を特殊トークン [MASK] に置き換える．そして，置き換えられた文章を BERT に入力し，[MASK] トークンの位置に元々あったトークンを予測する．

Next Sentence Prediction は二つの文章の関連性を理解するためのタスクである．事前学習には常に二つの文のペアが入力される．データの 50% は連続する文になっており，残りの 50% は連続していない文になっている．そして，2 つの文が連続しているかどうかを判定するタスクを用いて学習する．

2.3.5　fine-tuning

fine-tuning では解きたいタスクのラベル付きデータから，BERT がそのタスクに特化するように学習を行う．fine-tuning を行うときには，モデルのパラメータの初期値として，BERT のパラメータは事前学習で得られたパラメータを用い，新たに加えられた分類器のパラメータにはランダムな値を与える．そして，ラベル付きデータを用いて BERT と分類器の両方のパラメータを学習する．事前学習で得られたパラメータを初期値として使うことで，比較的少数の学習データからでも高い性能のモデルを得ることができる．

事前学習と fine-tuning をまとめて転移学習 (Transfer Learning) と呼ばれる．一連の流れは図 2.4 のようになる．

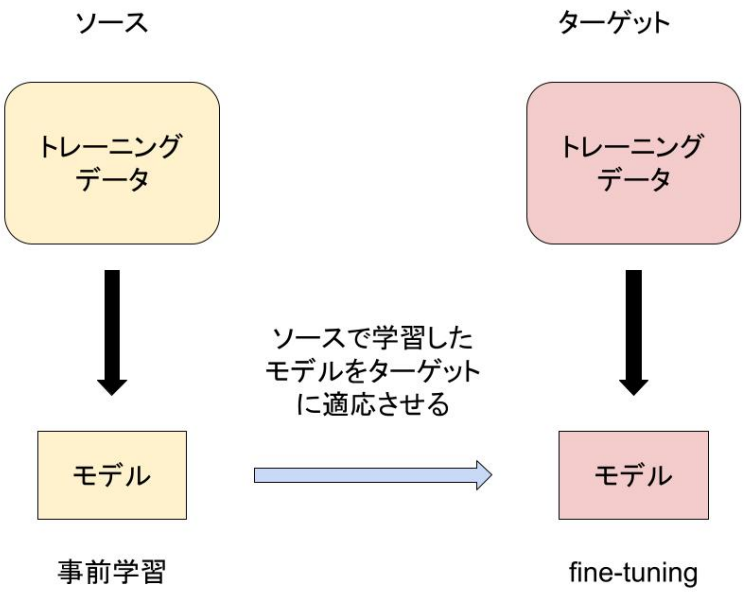

図 2.4: 転移学習の流れ.

2.4　RoBERTa

RoBERTa [8] [9] は BERT [1] のハイパーパラメータ，事前学習手法，埋め込み手法を分析することで，BERT と同じモデルサイズで下流タスクに対する精度を向上させたモデルである．BERT の事前学習は MLM と NSP である．

BERT のマスク手法はどのエポックでも毎回同じマスク (Static Masking) だったが，この論文ではモデルにテキストを入力するごとにマスキングする手法 (Dynamic Masking) を提案している．この手法では，エポック毎に違うマスキングが行われる．Dynamic Masking は学習回数が多かったり学習データが大きい場合に，重要である．

NSP では以下の 4 つの手法で比較実験を行い，NSP がモデルに与える影響を調べた．

- 複数の文章から構成されるセグメントのペアで入力文を作成する手法
- 2 つの文章によって入力文を作成する手法．トークン長が 512 に満たないため，バッチサイズを増やすことで対応した．
- 1 つか複数の文書から文章を抽出し，入力を構成する．文章を抽出する際，文書をまたぐ場合がある．その場合は，セパレーショントークンを挿入する．
- 1 つの文書から文章を抽出する手法．トークン長が 512 に満たないため，動的に

バッチサイズを変更することによって対応している．

実験結果から，NSP は性能向上にあまり寄与しないことが分かった．

Tect Encoding 手法としては，BERT は文字単位での分割を行っていたが，RoBERTa では入力を文字単位に変換した後バイト変換を行い，変換したバイトを用いて BPE のアルゴリズムを用いている．バイト単位は文字単位より性能が低下したが，論文では一貫した実験のためこの手法を用いている．

2.5　Multilingual BERT

Multilingual BERT(mBERT) は BERT と同じモデルアーキテクチャと学習方法を持っている多言語の事前学習済みモデル [1] である．mBERT が事前学習に用いる Wikipedia のデータには 104 の言語が含まれている．mBERT では WordPiece モデリングにより，モデルが言語間で埋め込み表現を共有することができている．語彙には様々な言語のキャラクタが入っており，語彙数は約 12 万 token ある．日本語の漢字は 1 文字で 2 文字の漢字単語はない．日本語のの 2 文字以上の token は約 1000 個ほどで，「になっている」,「となっていた」などの長いものもある．

Cross-Lingual な表現学習をしたモデルとして Goyal らの XLM-RoBERTa [10] がある．XLM-RoBERTa は様々なクロスリンガルベンチマークにおいて mBERT より大幅に優れている．また，XLM-RoBERTa は低リソース言語において特に優れた性能を示している．従来の XLM モデルと比較して XNLI の精度がスワヒリ語やウルドゥー語で向上している．このような，結果を得るためには

- 正の伝達と容量の希薄化
- 高リソース言語と低リソース言語のスケール

のトレードオフが性能に関係している．さらに，XLM-RoBERTa は GLUE と XLNI において強力な単言語モデルと非常に高い競争力を有している．

2.6　Zero Shot Cross-Lingual Transfer

Zero Shot Cross Lingual Transfer は Single-Source Transfer とも呼ばれ，ソース言語 (多くの場合，高リソース言語) でモデルを訓練し，その後ターゲット言語へ直接転送

する手法 [4] である．

近年の BERT 等の事前学習済みモデルは，言語をまたいだ転移学習が可能であることが知られている．例えば英語やフランス語などの間には，似た語彙も多く，英語で学習した知識がフランス語のタスクに有用なのは当然のことである．さらに，この転移学習は事前学習用の言語 (L1) と微調整用の言語 (L2) との間に，共通の語彙が全く無くても可能である．この結果から，言語をまたいだ転移学習には人間の言語の何らかの構造的特徴が関連していると考えられている．

Li らの事前学習された言語モデルにおける新たな cross-lingual 構造 [11] によると，多言語エンコーダの最上層に共有パラメータが存在するため，単言語コーパス間で共有する語彙がない場合やテキストが非常に異なる領域のものであっても Zero Shot Cross Lingual Transfer が可能であると示されている．

Artexte らの単言語表現の言語伝達可能性について [12] では，言語 L1 で事前学習された単言語モデルを言語 L2 コーパスを利用し新しいトークンの埋め込みを学習することにより新しい言語 L2 に転送した (共有サブワードの概念がない) モデルは最先端の多言語モデルと Zero Shot Cross Lingual Transfer ベンチマークで同等に機能した．これは，多言語モデルでは語彙の共有も共同事前トレーニングも必要ないことを示している．

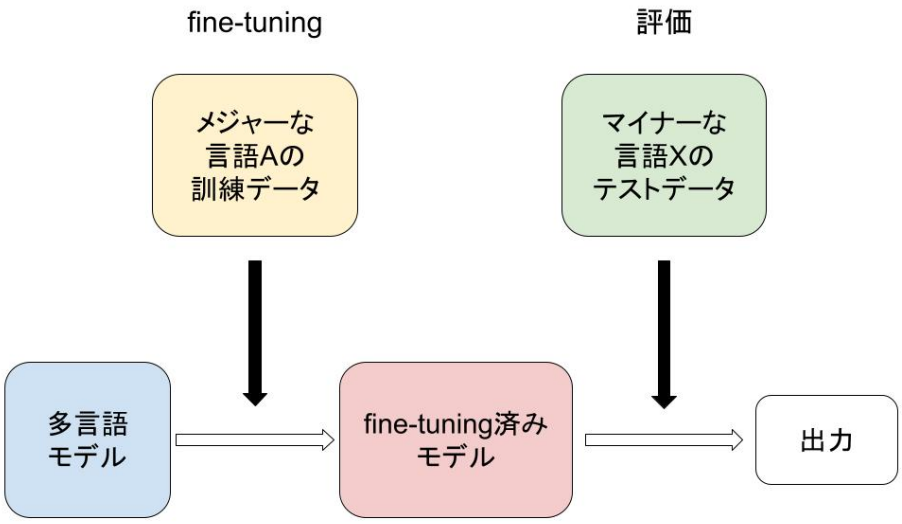

図 2.5: Zero Shot Cross Lingual Transfer の例.

つまり，図 2.5 のようにメジャーな言語 A で fine-tunig をしたモデルを，訓練データ

に含まれていないマイナーな言語 X のテストデータで評価をすることができる．

2.7　LLama 3.1 Swallow

Llama 3.1 Swallow [6] は，Meta 社の Llama 3.1 を基盤に，日本語能力を強化した大規模言語モデルである．このモデルは，英語の性能を維持しつつ，日本語の理解・生成能力を向上させることを目的としている．モデルサイズは 8B と 70B が提供されている．Llama 3.1 Swallow は，日本語の理解・生成タスクにおいて高い性能を示している．特に，指示チューニングを施したモデル（Instruct モデル）は，日本語のマルチターン対話能力が向上し，日本語 MT-Bench の平均スコアで 13B 以下の日本語 LLM の中ではトップクラスの成績を収めている．

2.8　Chain of Thought

Wei ら [13] は，言語モデルが複雑な推論タスクを解く際の新しいアプローチとして，Chain of Thought (CoT) Prompting を提案している．この手法は，単に結論を出力するのではなく，問題解決のプロセスを逐次的に記述する形式を言語モデルに促すものである．特に，数学問題や常識推論タスクなど，複数ステップの推論が必要な場面において，CoT が性能を大幅に向上させることを実証した．

また，GPT-3 [5] のような大規模言語モデルに CoT プロンプトを適用した際，従来のプロンプト方式と比較して顕著な精度向上が観察された．また，CoT の効果はモデルサイズに強く依存しており，100B パラメータ以上の大規模モデルで特に効果的であることが示された．

さらに，CoT プロンプトは特定のデータセットやタスクに依存することなく，汎用的に使用可能である点も強調されている．この手法は，推論能力を引き出すだけでなく，プロセスの透明性や説明可能性を向上させる点で注目される．一方で，モデルサイズが小さい場合や，タスクが単純すぎる場合には効果が限定的であるという課題も指摘されている．

また，Kojima ら [14] は Chain of Thought (CoT) を Zero Shot で適用する手法を提案し，大規模言語モデルが複雑な推論タスクを効率的に解けることを示した．この研究では，CoT を活用して問題解決のプロセスを逐次的に展開することで，言語モデルの推

論能力を向上させている.

第 3 章

提案手法

Zero Shot Cross Lingual Transfer を行う場合はメジャーな言語 A のデータとマイナーな言語 B のデータと多言語モデルを用いて行う．しかし，言語 A から言語 B の翻訳機と言語 B の単言語モデルがあれば，言語 A の訓練データを言語 B に翻訳し，翻訳したデータから言語 B の単言語モデルを fine-tuning し，その fine-tuning 済みモデルを利用して言語 B のテストデータを解析できる．このように，翻訳機が利用できれば単言語モデルであっても Zero Shot Cross Lingual Transfer と同等の処理が可能になる．この提案手法のイメージ図を図 3.1 に示す．

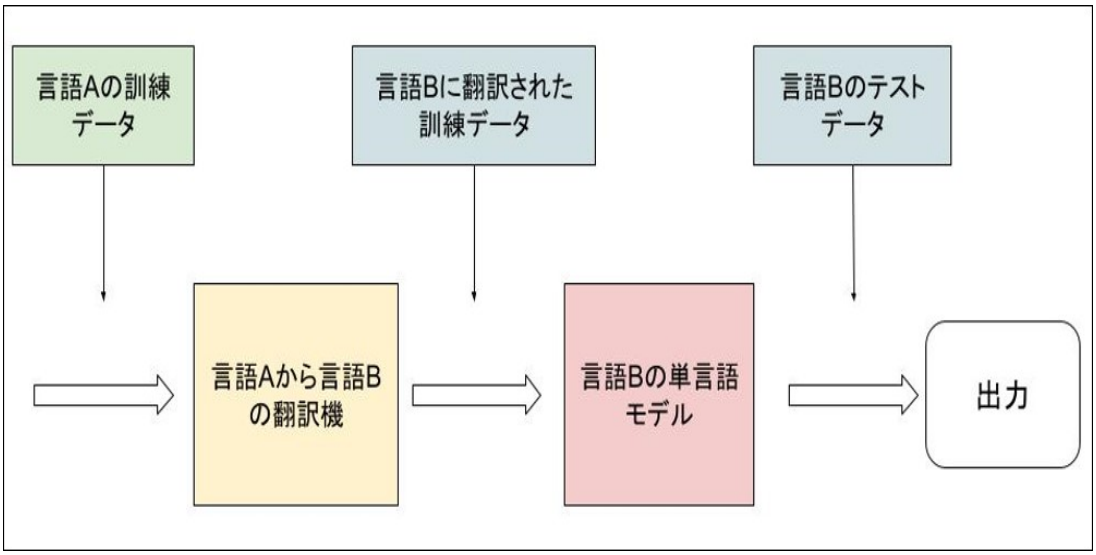

図 3.1: 提案手法のイメージ図.

しかし，多言語モデルの利用と MT + 単言語モデルの利用のどちらが有効であるかは明らかではないため，本稿では言語 A を英語，言語 B を日本語，タスクを Amazon レ

ビューの評判分析として以下の 3 つの手法の性能比較の実験を行った.

- (手法 1)xlm-RoBERTa を訓練データである英文書で fine-tuning し，テストデータである和文書でテストする (Zeroshot Cross-Lingual Transfer)
- (手法 2) 訓練データである英文書を翻訳機で和訳し，その和訳文書で日本語 RoBERTa を fine-tuning し，テストデータである和文書でテストする (MT ＋単言語)
- (手法 3) 訓練データである英文書を翻訳機で和訳し，その和訳文書ともとの英文書を合わせたデータで xlm-RoBERTa を fine-tuning し，テストデータである和文書でテストする (MT+ 多言語)

手法 1，手法 2，手法 3 それぞれの手法のイメージ図を図 3.2，図 3.3，図 3.4 に示す.

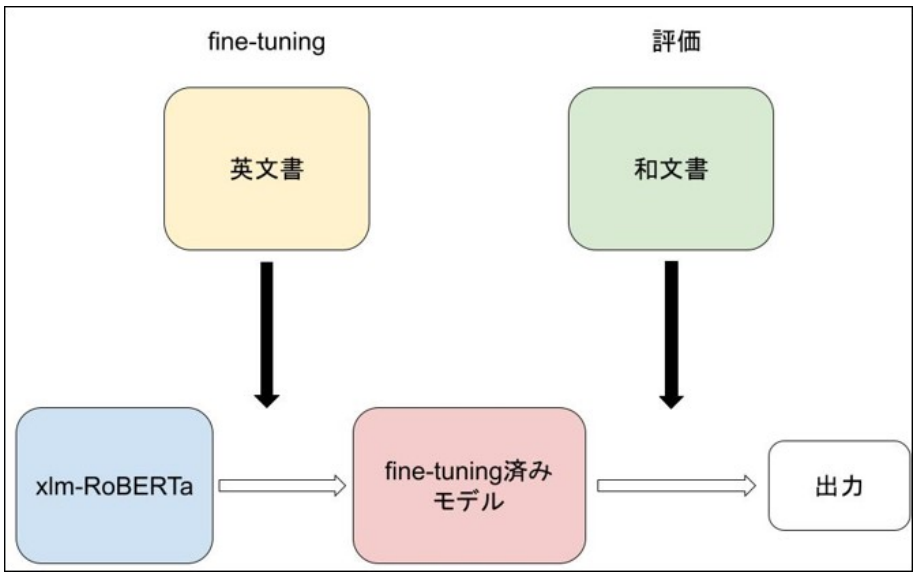

図 3.2: 手法 1 のイメージ図.

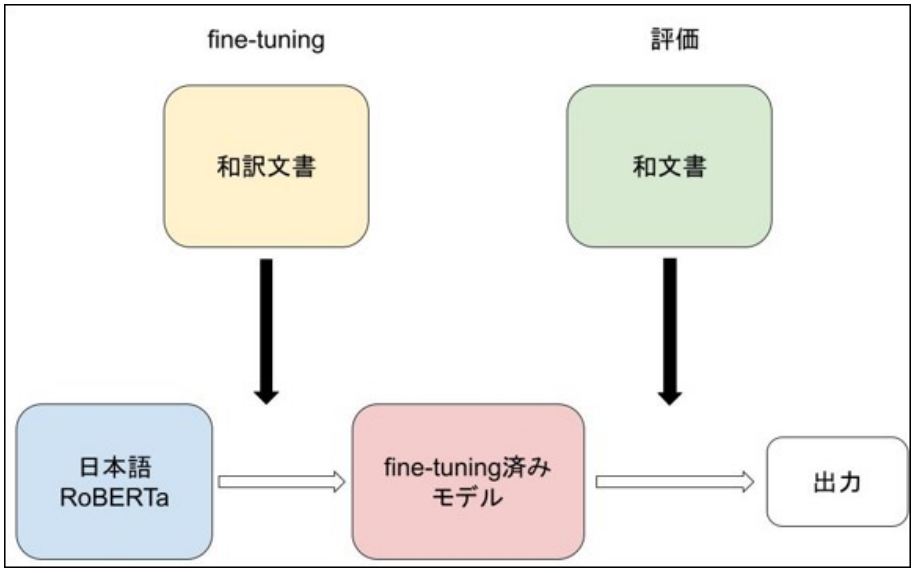

図 3.3: 手法 2 のイメージ図.

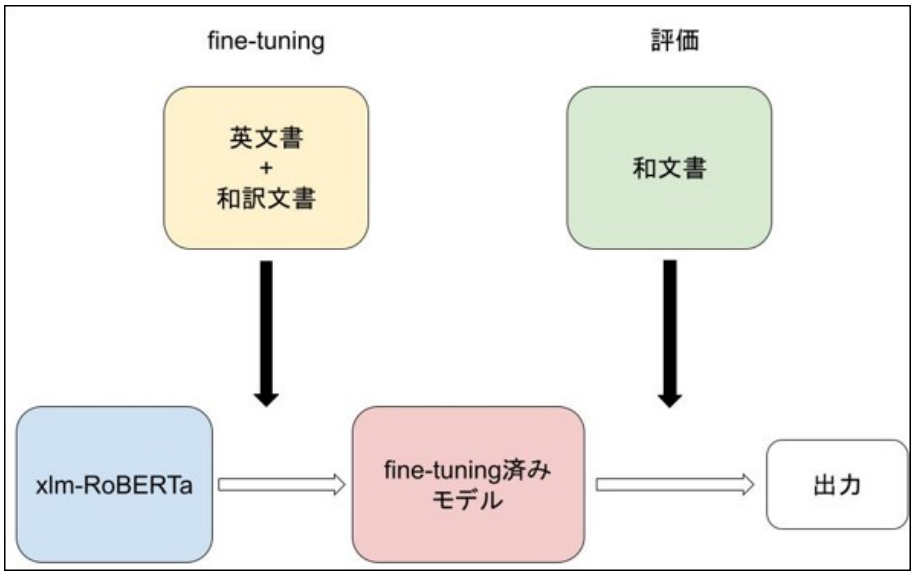

図 3.4: 手法 3 のイメージ図.

第 4 章

実験

Amazon レビューの評判分析を Zero Shot Cross Lingual Transfer で行う場合の多言語モデルの利用と MT + 単言語モデルの利用のどちらが有効かを確認した.

4.1 実験設定

4.1.1 事前学習済みモデル

多言語モデルは Conneau らによって公開された xlm-RoBERTa-base モデルを使用した．これは, Hugging Face 社の Transformers ライブラリ[*1]から, モデル名 'xlm-roberta-base' で利用できるモデルである．xlm-RoBERTa は RoBERTa の多言語版で，100 言語を含む 2.5TB のフィルター処理された CommonCrawl データで事前トレーニングされている．トークナイザーには SentencePiece を用いている．単言語モデルは rinnna 社から公開された日本語 RoBERTa を使用した．これは，Hugging Face 社の Transformers ライブラリからモデル名'rinna/japanese-roberta-base' で利用できるモデルである.

4.1.2 実験用データセット

実験には Webis-CLS-10 データセット[*2] を用いた．ラベルはレビューの星の数であり，1 から 5 までの 5 段階評価である．ただし，ラベルが 3 のデータは存在しない．本実験では，ラベルが 1，2 のデータをネガティブ，4，5 のデータをポジティブとして評

[*1] https://github.com/huggingface/transformers
[*2] https://webis.de/data/webis-cls-10.html

判分析（2 値分類）を行った．このデータセットには，日本語，英語，ドイツ語，フランス語それぞれに books，dvd，music の 3 つの領域がある．本実験では，すべて日本語の music の領域のデータを用いた．

また考察では，自作のテストデータとして，2024 年 7 月以降の Amazon レビューを使用した．

さらに，livedoor ニュース記事分類の実験では，NHN Japan 株式会社が運営する「livedoor ニュース」の記事部分のみを使用した．livedoor ニュースのカテゴリは，トピックニュース，Sports Watch，IT ライフハック，家電チャンネル，MOVIE ENTER，独女通信，エスマックス，livedoor HOMME，Peachy の 9 つである．

4.1.3　文書分類器の作成

手法 1 では訓練データの英文書 1600 文書と検証データの 400 文書で fine-tuning を行った．手法 2 では Google の機械翻訳で和訳した訓練データの和訳文書 1600 文書と検証データの和訳文書 400 文書で fine-tuning を行った．手法 3 では Google の機械翻訳で和訳した訓練データの和訳文書 16000 文書と英文書 1600 文書を合わせた 3200 文書と検証データの和訳文書 400 文書と英文書 400 文書を合わせた 800 文書で fine-tuning を行った．

4.1.4　fine-tuning する時のパラメータ

RoBERTa を fine-tuning をする時のパラメータはすべて同じにして実験を行った．主なパラメータを以下に示す．

- バッチサイズ:32
- 学習率:5e-5
- 最大エポック数:50
- patience:5

4.2　実験結果

英文書 2000 文で xlm-RoBERTa を fine-tuning したモデル (手法 1) と和訳文書 2000 文書で日本語 RoBERTa を fine-tuning したモデル (手法 2) と英文書と和訳文書を合わせた 4000 文書で xlm-RoBERTa を fine-tuning したモデル (手法 3) をテストデータである和文書 300 文書で評判分析を行い精度の比較をした．正解率は 5 回の平均値とした．評判分析の正解率を表 4.1 に示す．また，実験結果のグラフを図 4.1 に示す．実験結果より，多言語の Zero Shot Cross Lingual Transfer における評判分析 (手法 1) の正解率よりも MT を利用した単言語モデルの評判分析 (手法 2) の方が正解率が高かった．また，MT を利用した多言語モデルの評判分析 (手法 3) の正解率が最も低いという結果になった．

表 4.1: 和文書で評価した正解率

訓練データとモデル	日本語で評価
手法 1	0.812
手法 2	0.87
手法 3	0.8013

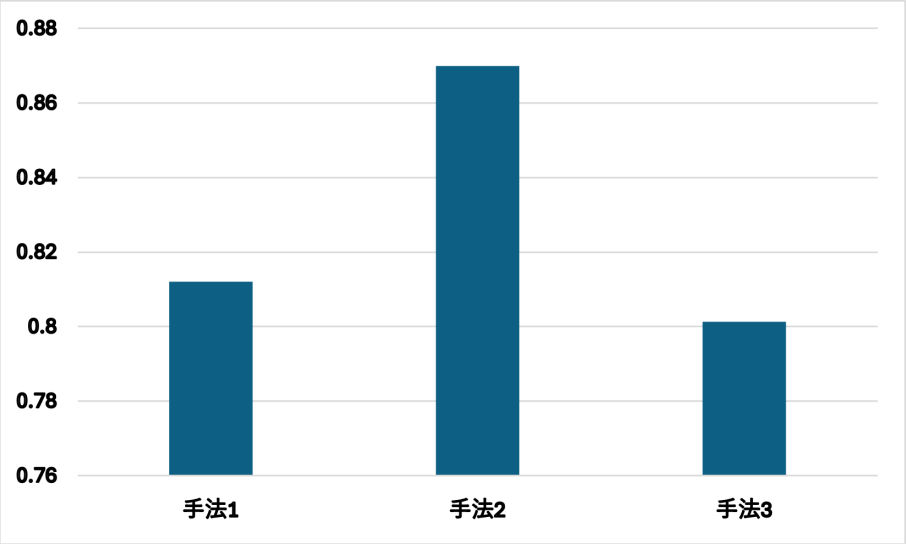

図 4.1: 実験結果の正解率のグラフ．

第 5 章

考察

LLM を利用した Zero Shot 評判分析の性能を GPT3.5-turbo，GPT-4o，Llama 3.1-Swallow-8B を用いて，4 章と同じデータセット，同じタスクで調査した．

5.1 実験設定

5.1.1 大規模言語モデル

LLM として GPT-3.5-turbo，GPT-4o，Llama 3.1-Swallow-8B を用いた．GPT-3.5-turbo と GPT-4o はそれぞれ OpenAI の API でモデル名をそれぞれ'gpt-3.5-turbo'，'gpt-4o' で利用できるモデルを使用した．Llama 3.1-Swallow-8B は Hugging Face 社の Transformers ライブラリから，モデル名'tokyotech-llm/Llama-3.1-Swallow-8B-Instruct-v0.1' で利用できるモデルを使用した．

また，実験の比較対象として，rinna 社の日本語単言語モデルを使用した．このモデルは，Hugging Face 社の Transformers ライブラリから，モデル名'rinna/japanese-roberta-base' として利用可能なものを使用した

5.1.2 データセット

4 章で用いたテストデータと同じく，Webis-CLS-10 データセットの日本語 300 文書をテストデータとした．

5.1.3　LLM に与えるプロンプト

OpenAI が公開している GPT の API を利用する場合や Swallow を利用する場合，2 つのプロンプトを渡すことで生成を行う．以下にその 2 つを示す．

- system: チャットボットの動作や指針を設定するメッセージ．モデルに指示を与える役割．
- user: ユーザーからの入力を表すメッセージ．ユーザーが提供する情報．

実験で使用するプロンプトはタスクに応じて変更しているが，基本的にはタスクを説明する文，ラベルの説明，および判断理由の出力の有無を含めたものとしている．また，LLM が明確な回答を出さない場合があるため，必ず回答するように指示を加えた．判断理由の出力については，LLM に理由を説明させることで性能が向上する「Chain of Thought」アプローチが日本語の Zero Shot 学習における評判分析において効果を持つかを検証する目的で含めている．一方，user では評判分析を行う文書のみを提示した．

LLM に対して学習データを与えずに対話形式でタスクを解くことで，LLM の Zero Shot 学習における精度やタスクの問題について調査を行う．以下の 5 つの条件で調査を実施した．

- 理由出力なし，300 文書の Amazon レビューの評判分析 (2 値分類) (実験 1)
- 理由出力あり，300 文書の Amazon レビューの評判分析 (2 値分類)(実験 2)
- 理由出力なし，自作の 100 文書の Amazon レビューの評判分析 (2 値分類)(実験 3)
- 理由出力なし，300 文書の Amazon レビューの評判分析 (4 値分類)(実験 4)
- 理由出力なし，900 文書の livedoor ニュース記事のカテゴリ分類 (実験 5)

5.2　実験結果

実験 1 では，日本語の Amazon レビュー 300 文書を用いたテストデータに対して，理由を出力させながら GPT-3.5-turbo，GPT-4o，Swallow を使用してタスクを解いた．ポジティブおよびネガティブ以外の出力があった場合は，不正解として集計した．また，比較対象として日本語の RoBERTa を 1600 文書で学習した場合の正解率と手法 2 の正

解率も示す．それぞれの正解率を表 5.1 に示す．日本語の RoBERTa の正解率は 5 回の実験結果の平均値を使用した．また，監視する指標を loss と Accuracy の 2 パターンで評価を行った．

表 5.1: Amazon レビュー 300 文書の理由出力ありの正解率

モデル	正解率
GPT-3.5-turbo	0.8567
GPT-4o	0.9433
Swallow	0.9
RoBERTa(loss)	0.8933
RoBERTa(Accuracy)	0.8879
手法 2	0.87

実験 2 では，日本語の Amazon レビュー 300 文書を用いたテストデータに対して，理由を出力させずに GPT-3.5-turbo，GPT-4o，Swallow を使用してタスクを解いた．また，比較対象として日本語の RoBERTa を 2000 文書で学習した場合の正解率と手法 2 の正解率を示す．それぞれの正解率を表 5.2 に示す．

表 5.2: Amazon レビュー 300 文書の理由出力なしの正解率

モデル	正解率
GPT-3.5-turbo	0.90
GPT-4o	0.9567
Swallow	0.9333
RoBERTa(loss)	0.8933
RoBERTa(Accuracy)	0.8879
手法 2	0.87

実験の結果，GPT-3.5-turbo において理由出力ありの場合は教師あり学習よりも精度が劣る結果となった．一方で，理由出力ありの場合の GPT-4o と Swallow，理由出力なしの場合の GPT-3.5-turbo および GPT-4o，Swallow は，Zero Shot 学習においても教師あり学習を上回る精度を達成した．

また，他言語の訓練データを用いて評判分析を行う Zero Shot Cross Lingual Transfer よりも，訓練データなしで LLM を利用した Zero Shot 学習による評判分析の方が精度が高いことが確認できた．また，理由出力を求めた場合，理由出力を求めない場合と比較して精度が低下することが確認された．LLM に理由出力を求める場合，モデルは予測精度に加えて理由生成の品質も同時に最適化する必要があり，この二重の負担が Zero Shot 学習におけるモデルの能力を分散させる原因となった．その結果，日本語の Amazon レビューの評判分析において適切な予測を行うことが難しくなったと考えられる．また，Amazon レビューの評判分析は言語推論が重要ではないタスクであるため，数学的推論のように CoT（Chain of Thought）が有効に機能しなかった可能性がある．そのため，これ以降の実験ではすべて理由出力を求めずに実施することとした．

Webis-CLS-10 データセットは 2010 年以前のデータであり，LLM の事前学習に含まれている可能性があるため，高い精度を示す結果となった可能性がある．そのため，2024 年 7 月以降の Amazon レビューを使用して，日本語 100 文書のテストデータを作成した．実験 3 では，最新のデータでも精度が高いのかを確認するため，作成した自前のデータセットで評判分析を行った．その結果を表 5.3 に示す．

表 5.3:　自作の Amazon レビュー 100 文書の理由出力なしの正解率

モデル	正解率
GPT-3.5-turbo	0.93
GPT-4o	0.97
Swallow	0.93

実験 3 の結果，最新のデータでも精度は変わらないことが確認できた．

Zero Shot 学習でもポジネガ判定では高い性能を示したが，より細かいクラス分けを行う 4 値分類でも同様に高い精度を維持できるかを検証する．実験 4 では，実験 1 と 2 で使用したテストデータの星の数を予測する 4 値分類として評判分析を行った．

実験 1 と 2 同様に，日本語の Amazon レビュー 300 文書を用いたテストデータで，理由出力を求めずに GPT-3.5-turbo，GPT-4o，Swallow を使用して解いた．また，比較対象として，同様に日本語の RoBERTa を 2000 文書で学習した場合の正解率とを示す．それぞれの正解率を表 5.4 に示す．

実験 4 の結果，4 値分類では精度が全体的に低下するものの，LLM の Zero Shot 学

表 5.4: Amazon レビュー 300 文書の 4 値分類の理由出力なしの正解率

モデル	正解率
GPT-3.5-turbo	0.60
GPT-4o	0.63
Swallow	0.5933
RoBERTa(loss)	0.47
RoBERTa(Accuracy)	0.588

習でも RoBERTa の教師あり学習と同等の精度を示すことがわかった.

極性判定のようにラベル間の境界がある程度わかりやすいタスクではなく，文書分類のようにラベル間の境界が曖昧なタスクでは，LLM の Zero Shot 学習における精度を検証するため，livedoor ニュース記事の文書分類を実験 5 として行った．実験 5 では，livedoor のニュース記事 900 文書を GPT-3.5-turbo，GPT-4o，Swallow で文書分類した．また，比較対象として日本語の RoBERTa を 540 文書で学習した場合の正解率も示す．それぞれの正解率を表 5.5 に示す.

表 5.5: livedoor ニュース記事 900 文書の文書分類の正解率

モデル	正解率
GPT-3.5-turbo	0.5389
GPT-4o	0.7089
Swallow	0.57
RoBERTa(loss)	0.6287
RoBERTa(Accuracy)	0.8529

実験 5 の結果，livedoor ニュース記事の文書分類タスクでは，Zero Shot 学習の LLM よりも教師あり学習の RoBERTa の方が精度が高くなった．この結果から，Zero Shot 学習における LLM は，ラベル間の境界が曖昧なタスクでは RoBERTa の教師あり学習よりも精度が下がることがわかった.

第 6 章

結論

本稿では翻訳機 MT が利用できる場合に多言語モデルを利用する方がよいのか，MT+ 単言語モデルを利用する方がよいのかを調べることを目的に，タスクを Amazon レビューとし手法 1(通常の Zero Shot Cross Lingual Transfer)，手法 2(MT+ 単言語モデル)，手法 3(MT ＋多言語モデル) で実験を行い調査した．実験の結果，手法 3 より手法 1 の性能が高く，手法 2 の性能が最も高かった．

また，LLM を学習データなしで利用し，Amazon レビューの評判分析および livedoor ニュース記事の文書分類タスクを解かせた．これにより，Zero Shot 学習の精度比較，出力理由の分析，ならびにタスクの違いによる精度の比較を行った．その結果，GPT-4o や Swallow は Zero Shot 学習でも極性判定において高い精度を示した一方で，文書分類タスクでは精度が低下する傾向が見られた．また，LLM を用いた Zero Shot の日本語の Amazon レビューの評判分析において，CoT（Chain of Thought）を導入すると，精度が低下することが確認された．また，他言語の訓練データを用いて評判分析を行う Zero Shot Cross Lingual Transfer よりも，訓練データなしで LLM を利用した Zero Shot 学習による評判分析の方が精度が高いことが確認できた．

今後の課題として，日本語の Zero Shot 学習における LLM の精度を向上させるプロンプトの開発や，LLM が Zero Shot 学習で得意とするタスクの特定について検討する．

謝辞

本研究を進めるにあたって，多くのご指導を頂いた指導教員の新納浩幸教授に感謝いたします．また，日常の議論を通して多くの知識，示唆を頂いた新納研究室の皆様に感謝します．

参考文献

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[2] Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining, 2019.

[3] Naman Goyal, Jingfei Du, Myle Ott, Giri Anantharaman, and Alexis Conneau. Larger-scale transformers for multilingual masked language modeling, 2021.

[4] Shijie Wu and Mark Dredze. Beto, bentz, becas: The surprising cross-lingual effectiveness of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 833–844, Hong Kong, China, November 2019. Association for Computational Linguistics.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[6] 産業技術総合研究所岡崎研究室. Llama 3.1 swallow. `https://swallow-llm.github.io/llama3.1-swallow.ja.html`, 2024. 2024-12 閲覧.

[7] Ashish Vaswani, NoamShazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[9] kryouhei673. Bert 派生モデルの論文紹介. `https://qiita.com/kryouhei673/items/cd6d5537ea13f1015146`. 閲覧日: 2021 年 12 月 2 日.

[10] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8440–8451, Online, July 2020. Association for Computational Linguistics.

[11] Alexis Conneau, Shijie Wu, Haoran Li, Luke Zettlemoyer, and Veselin Stoyanov. Emerging cross-lingual structure in pretrained language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6022–6034, Online, July 2020. Association for Computational Linguistics.

[12] Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. On the cross-lingual transferability of monolingual representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4623–4637, Online, July 2020. Association for Computational Linguistics.

[13] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[14] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023.

付録

A　実験で使用したソースコード

手法 1，手法 2，手法 3 で用いた Zero Shot Cross Lingual Transfer のソースコードを A.1 に示す.

ソースコード A.1: Zero Shot Cross Lingual Transfer の評判分析のソースコード

```
import os
import argparse
from pathlib import Path
import pathlib
from sre_constants import OP_IGNORE
from turtle import forward
import torch
from torch.utils.data import DataLoader, Dataset
from torch.optim.lr_scheduler import StepLR

import pytorch_lightning as pl
from pytorch_lightning.callbacks import EarlyStopping, ModelCheckpoint

from transformers import AdamW
from transformers import AutoTokenizer,
    AutoModelForSequenceClassification
import torchmetrics

from pytorch_lightning import loggers as pl_loggers
from pytorch_lightning import seed_everything

class ClassificationDataset(Dataset):#データセットの作成
    def __init__(self, data_file: Path, tokenizer: AutoTokenizer) ->
        None:
```

```
23          super().__init__()
24          self.input_ids = []
25
26          with open(data_file, 'r') as reader:
27              for line in reader:
28                  row = line.strip().split(',')
29                  if int(row[0]) < 3:#1，2のラベルを0（ネガティブ）に，
                        4，5のラベルを1（ポジティブ）にする
30                      l =  0
31                  else:
32                      l = 1
33                  label = l
34                  text  = ' '.join(row[1:])
35
36                  encoded = tokenizer.encode_plus(
37                      text,
38                      add_special_tokens=True,
39                      max_length = 255,
40                      padding='max_length',
41                      truncation=True,
42                      return_attention_mask=True,
43                      return_tensors='pt'
44                  )
45
46                  self.input_ids.append(
47                      dict(
48                          input_ids=encoded['input_ids'].flatten(),
49                          attention_mask=encoded['attention_mask'].
                                flatten(),
50                          labels=torch.tensor(label)
51                      )
52                  )
53
54      def __len__(self) -> int:
55          return len(self.input_ids)
56
57      def __getitem__(self, index) -> dict:
58          return self.input_ids[index]
59
60
61  class ClassificationDataModule(pl.LightningDataModule):
```

```
62      def __init__(self, dataset_dir: Path, batch_size : int,
            pretrained_model_name: str) -> None:
63          super().__init__()
64          #データの指定を変更することによって手法が変わる
65          self.train_file = dataset_dir.joinpath('train_en_trans.csv')#訓
                練データの指定
66          self.val_file = dataset_dir.joinpath('val_en_trans.csv')#検証
                データの指定
67          self.test_file = dataset_dir.joinpath('ja_test300.csv')#テスト
                データの指定
68          self.batch_size = batch_size
69          self.tokenizer = AutoTokenizer.from_pretrained(
                pretrained_model_name)
70          self.tokenizer.do_lower_case = True
71
72      def setup(self, stage = None) -> None:
73          self.train_dataset = ClassificationDataset(self.train_file,
                self.tokenizer)
74          self.val_dataset = ClassificationDataset(self.val_file, self.
                tokenizer)
75          self.test_dataset = ClassificationDataset(self.test_file, self
                .tokenizer)
76
77      def train_dataloader(self) -> DataLoader:
78          return DataLoader(self.train_dataset, batch_size=self.
                batch_size, shuffle=True, num_workers=os.cpu_count(),
                pin_memory=True)
79
80      def val_dataloader(self) -> DataLoader:
81          return DataLoader(self.val_dataset, batch_size=self.batch_size,
                 shuffle=False, num_workers=os.cpu_count(), pin_memory=
                True)
82
83      def test_dataloader(self) -> DataLoader:
84          return DataLoader(self.test_dataset, batch_size=self.batch_size
                , shuffle=False, num_workers=os.cpu_count(), pin_memory=
                True)
85
86
87  class Classifier(pl.LightningModule):#分類期の作成
88      def __init__(self, bert_model: AutoModelForSequenceClassification)
            -> None:
```

```
89          super().__init__()
90          self.model = bert_model
91
92          self.metric = torchmetrics.Accuracy(task="binary")
93
94          self.training_epoch_outputs = []
95
96      def forward(self, input_ids, attention_mask, labels=None):
97          outputs = self.model(input_ids, attention_mask=attention_mask,
                labels=labels)
98          return outputs['loss'], outputs['logits']
99
100     def training_step(self, batch, batch_idx):
101         loss, preds = self.forward(input_ids=batch["input_ids"],
102                                    attention_mask=batch["
                                       attention_mask"],
103                                    labels=batch["labels"])
104         return {'loss': loss,
105                 'batch_preds': preds,
106                 'batch_labels': batch["labels"]}
107
108     def on_train_epoch_end(self):
109         epoch_loss = torch.sum(torch.tensor([x['loss'] for x in self.
                training_epoch_outputs]))
110         self.log(
111             'train_loss',
112             epoch_loss,
113             prog_bar=True,
114             logger=True,
115             on_epoch=True
116         )
117         self.training_epoch_outputs = []
118
119     def validation_step(self, batch, batch_idx):
120         loss, preds = self.forward(input_ids=batch["input_ids"],
121                                    attention_mask=batch["
                                       attention_mask"],
122                                    labels=batch["labels"])
123         return {'loss': loss,
124                 'batch_preds': preds,
125                 'batch_labels': batch["labels"]}
```

```
126
127     def on_validation_epoch_end(self):
128         epoch_loss = torch.sum(torch.tensor([x['loss'] for x in self.
                training_epoch_outputs]))
129         self.log(
130             'valid_loss',
131             epoch_loss,
132             prog_bar=True,
133             logger=True,
134             on_epoch=True
135         )
136         self.training_epoch_outputs = []
137
138     def test_step(self, batch, batch_idx):
139         loss, preds = self.forward(input_ids=batch["input_ids"],
140                                    attention_mask=batch["
                                        attention_mask"],
141                                    labels=batch["labels"])
142         return {'loss': loss,
143                 'batch_preds': preds,
144                 'batch_labels': batch["labels"]}
145
146     def on_test_epoch_end(self) -> None:
147         epoch_loss = torch.sum(torch.tensor([x['loss'] for x in self.
                test_results]))
148         epoch_preds = torch.cat([x['batch_preds'] for x in self.
                test_results])
149         epoch_labels = torch.cat([x['batch_labels'] for x in self.
                test_results])
150
151         epoch_preds = torch.argmax(epoch_preds, dim=1)
152
153         print(epoch_preds)
154         self.metric(epoch_preds, epoch_labels)
155         self.log(
156             'test_loss',
157             epoch_loss,
158             prog_bar=True,
159             logger=True,
160             on_epoch=True
161         )
```

```
162         self.log(
163             'test_accuracy',
164             self.metric,
165             prog_bar=True,
166             logger=True,
167             on_epoch=True
168         )
169
170     def test_step(self, batch, batch_idx):
171         loss, preds = self.forward(input_ids=batch["input_ids"],
172                                    attention_mask=batch["
                                           attention_mask"],
173                                    labels=batch["labels"])
174         return {'loss': loss,
175                 'batch_preds': preds,
176                 'batch_labels': batch["labels"]}
177
178     def on_test_start(self):
179         self.test_results = []
180
181     def test_step(self, batch, batch_idx):
182         loss, preds = self.forward(input_ids=batch["input_ids"],
183                                    attention_mask=batch["
                                           attention_mask"],
184                                    labels=batch["labels"])
185         results =  {'loss': loss,
186                 'batch_preds': preds,
187                 'batch_labels': batch["labels"]}
188         self.test_results.append(results) # 結果をリストに追加
189         return results
190
191     def configure_optimizers(self):
192         optimizer = AdamW(self.model.parameters(), lr=5e-5)
193         scheduler = {'scheduler': StepLR(optimizer=optimizer,
                step_size=1, gamma=0.5)}
194         return [optimizer]
195
196 if __name__ == '__main__':
197     parser = argparse.ArgumentParser()
198     parser.add_argument('--seed', type=int, default=600, help='seed')
199     args = parser.parse_args(args=[])
```

```
200
201     seed=args.seed
202     seed_everything(seed=seed, workers=True)
203     num_epoch = 50#エポック数
204
205     pretrained_model_name = 'rinna/japanese-roberta-base'#事前学習済み
            モデルの指定
206     bert_model = AutoModelForSequenceClassification.from_pretrained(
            pretrained_model_name, num_labels=2)
207     model = Classifier(bert_model)
208
209     dataset_dir = pathlib.Path('dataset', 'trans')
210     batch_size = 32#バッチサイズ
211
212     run_name = 'task3'
213
214     early_stop_callback = EarlyStopping(
215         monitor='valid_loss',
216         min_delta=0.01,
217         patience=5,
218         mode='min'
219     )
220
221     checkpoint_callback = ModelCheckpoint(
222         dirpath="./checkpoints/{}-seed{}".format(run_name, seed),
223         filename='{epoch}',
224         verbose=True,
225         monitor='valid_loss',
226         mode='min'
227     )
228
229     trainer = pl.Trainer(
230     max_epochs=num_epoch,
231     accelerator='gpu',
232     devices=1,
233     callbacks=[checkpoint_callback, early_stop_callback],
234     logger=pl_loggers.TensorBoardLogger(save_dir='logs/{}/'.format(
            run_name))
235     )
236
237     data_module = ClassificationDataModule(dataset_dir=dataset_dir,
```

```
            batch_size=batch_size, pretrained_model_name=
            pretrained_model_name)
238
239     trainer.fit(model=model, datamodule=data_module)
240     #正解率の表示
241     result = trainer.test(ckpt_path=checkpoint_callback.best_model_path
            , datamodule=data_module)
```

また，OpenAI の API を利用して Zero Shot で評判分析するソースコードを A.2 に示す.

ソースコード A.2: OpenAI の API を利用して Zero Shot で評判分析するソースコード

```
1  import openai
2  import pandas as pd
3  import time
4  #取得した API キーを入力
5  api_key = "API_KEY"
6
7  openai.api_key = api_key
8
9  input_file = 'dataset/trans/ja_test300.csv'
10 output_file = 'dataset/trans/gpt4o_300_noreason.csv'
11
12 df = pd.read_csv(input_file, header=None, names=["Label", "Text"])
13
14 responses = []
15
16 for index, row in df.iterrows():
17     prompt = row["Text"]
18     print(f"Processing row {index + 1}/{len(df)}...")
19     try:
20         response = openai.ChatCompletion.create(
21             model="gpt-4o",#使用するモデルの指定
22             messages=[
23                 # LLM に渡すプロンプト
24                 {"role": "system", "content": "あなたは
                       Amazon レビューの文書を評判分析する AI です。
25                 文書の星の数が 4以上だと思うなら「ポジティブ」、2以下だ
                       と思うなら「ネガティブ」と出力してください。そう思
                       う理由などは必要ありません。
26                 理解不能や意味不明の文書でも必ず「ネガティブ」か「ポジ
```

```
                    ティブ」のどちらかを出力してください。"},
                {"role": "user", "content": prompt}
            ]
        )
        answer = response.choices[0].message['content']
    except Exception as e:
        answer = f"Error: {e}"

    responses.append([index + 1, row["Label"], answer])

    time.sleep(1)

output_df = pd.DataFrame(responses, columns=["Index", "Label", "
    Response"])

output_df.to_csv(output_file, index=False, header=True)

print(f"処理が完了しました。結果は
    '{output_file}' に保存されています。")
```

また，Swallow を利用して Zero Shot で評判分析するソースコードを A.3 に示す

ソースコード A.3: Swallow を利用して Zero Shot の評判分析をするソースコード

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch
from transformers import TextIteratorStreamer
from threading import Thread
import csv

# トークナイザーとモデルの準備
tokenizer = AutoTokenizer.from_pretrained(
    "tokyotech-llm/Llama-3.1-Swallow-8B-Instruct-v0.1"
)
model = AutoModelForCausalLM.from_pretrained(
    "tokyotech-llm/Llama-3.1-Swallow-8B-Instruct-v0.1",
    device_map="cuda",
    torch_dtype="auto",
    load_in_8bit=True,
)

input_file = "dataset/trans/ja_test300.csv"
```

```
output_file = "dataset/trans/swallow_300_noreason.csv"
# LLMに渡すプロンプト
DEFAULT_SYSTEM_PROMPT = "あなたは
    Amazonレビューの文書を評判分析するAIです。
文書の星の数が4以上だと思うなら「ポジティブ」、2以下だと思うなら「ネガ
    ティブ」と出力してください。そう思う理由などは必要ありません。
理解不能や意味不明の文書でも必ず「ネガティブ」か「ポジティブ」のどちら
    かを出力してください。"

with open(input_file, newline='', encoding='utf-8') as csvfile, open(
    output_file, 'w', encoding='utf-8') as outfile:
    reader = csv.reader(csvfile)
    writer = csv.writer(outfile)

    writer.writerow(["Index", "Label", "Response"])

    for line_num, row in enumerate(reader, start=1):
        label = row[0]
        text = row[1]

        chat = [
            {"role": "system", "content": DEFAULT_SYSTEM_PROMPT},
            {"role": "user", "content": text}
        ]
        prompt = tokenizer.apply_chat_template(chat, tokenize=False,
            add_generation_prompt=True)

        # 推論の実行
        input_ids = tokenizer.encode(prompt, add_special_tokens=False,
             return_tensors="pt").to(model.device)
        streamer = TextIteratorStreamer(tokenizer, skip_prompt=True,
            skip_special_tokens=True)

        def generate():
            model.generate(
                input_ids,
                max_new_tokens=512,
                streamer=streamer
            )

        thread = Thread(target=generate)
```

```
53          thread.start()
54
55          generated_text = ""
56          for chunk in streamer:
57              print(chunk, end="", flush=True)
58              generated_text += chunk
59
60          thread.join()
61
62          writer.writerow([line_num, label, generated_text])
```