Handout 2: BERT for Sentiment Classification (Intermediate Level)

Machine Learning for Smarter Innovation

1 Handout 2: BERT for Sentiment Classification (Intermediate Level)

1.1 Week 3 - Advanced NLP with Transformers

1.1.1 Understanding BERT (Bidirectional Encoder Representations from Transformers)

BERT revolutionized NLP by reading text bidirectionally, understanding context from both left and right simultaneously. This enables much more nuanced sentiment analysis than traditional methods.

1.1.2 Transformer Architecture Basics

Key Components

1. **Self-Attention**: Allows the model to weigh the importance of different words
2. **Positional Encoding**: Helps the model understand word order
3. **Multi-Head Attention**: Captures different types of relationships
4. **Feed-Forward Networks**: Processes the attended information

```
# BERT processes text like this:
# Input: "The product is not bad"
# Traditional (left-to-right): "not"     "bad" = negative
# BERT (bidirectional): "not bad" in context = positive
```

1.1.3 Setting Up BERT for Sentiment Analysis

Installation

```
pip install transformers torch pandas numpy
```

Basic Implementation

```
from transformers import pipeline, AutoTokenizer,
    AutoModelForSequenceClassification
import torch

# Method 1: Using Pipeline (Easiest)
def setup_bert_pipeline():
    # Load pre-trained sentiment analysis model
    classifier = pipeline(
        "sentiment-analysis",
        model="nlptown/bert-base-multilingual-uncased-sentiment"
    )
    return classifier

# Usage
classifier = setup_bert_pipeline()
result = classifier("This product exceeded my expectations!")
print(result)  # [{'label': '5 stars', 'score': 0.89}]
```

Custom Model Loading

```
# Method 2: More Control
def setup_custom_bert():
    model_name = "bert-base-uncased"
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForSequenceClassification.from_pretrained(
        model_name,
        num_labels=3  # positive, negative, neutral
    )
    return tokenizer, model

def predict_sentiment(text, tokenizer, model):
    # Tokenize input
    inputs = tokenizer(
        text,
        return_tensors="pt",
        truncation=True,
        padding=True,
        max_length=512
    )

    # Get predictions
    with torch.no_grad():
        outputs = model(**inputs)
        predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)

    # Convert to labels
    labels = ['negative', 'neutral', 'positive']
    scores = predictions.numpy()[0]

    return {label: float(score) for label, score in zip(labels, scores)}
```

1.1.4 Using Pre-trained Models

Available Models for Sentiment

```
# Different pre-trained models for various needs
models = {
    'general': 'cardiffnlp/twitter-roberta-base-sentiment',
    'finance': 'ProsusAI/finbert',
    'multilingual': 'nlptown/bert-base-multilingual-uncased-sentiment',
    'emotion': 'j-hartmann/emotion-english-distilroberta-base'
}

def load_domain_model(domain='general'):
    model_name = models.get(domain, models['general'])
    return pipeline("sentiment-analysis", model=model_name)

# Example: Finance-specific sentiment
finance_classifier = load_domain_model('finance')
result = finance_classifier("Stock prices soared after earnings beat
    expectations")
```

1.1.5 Fine-tuning BERT for Custom Domains

Preparing Your Data

```
import pandas as pd
from sklearn.model_selection import train_test_split

def prepare_data(csv_file):
    # Load your labeled data
    df = pd.read_csv(csv_file)

    # Expected columns: 'text', 'label'
    # Labels: 0=negative, 1=neutral, 2=positive

    # Split data
    X_train, X_val, y_train, y_val = train_test_split(
        df['text'].tolist(),
        df['label'].tolist(),
        test_size=0.2,
        random_state=42
    )

    return X_train, X_val, y_train, y_val
```

Fine-tuning Process

```
from transformers import TrainingArguments, Trainer
from torch.utils.data import Dataset
import torch

class SentimentDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_length=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = str(self.texts[idx])
        label = self.labels[idx]

        encoding = self.tokenizer(
            text,
            truncation=True,
            padding='max_length',
            max_length=self.max_length,
            return_tensors='pt'
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }

def fine_tune_bert(X_train, y_train, X_val, y_val):
    # Load pre-trained model
    model_name = "bert-base-uncased"
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForSequenceClassification.from_pretrained(
        model_name,
        num_labels=3
    )

    # Create datasets
    train_dataset = SentimentDataset(X_train, y_train, tokenizer)
    val_dataset = SentimentDataset(X_val, y_val, tokenizer)

    # Training arguments
    training_args = TrainingArguments(
        output_dir='./results',
        num_train_epochs=3,
        per_device_train_batch_size=16,
        per_device_eval_batch_size=64,
        warmup_steps=500,
        weight_decay=0.01,
        logging_dir='./logs',
        evaluation_strategy="epoch",
        save_strategy="epoch",
        load_best_model_at_end=True,
    )

    # Create trainer
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=val_dataset,
    )

    # Train
    trainer.train()

    return model, tokenizer
```

1.1.6 Evaluation Metrics and Validation

Key Metrics

```
from sklearn.metrics import accuracy_score, precision_recall_fscore_support,
    confusion_matrix
import numpy as np

def evaluate_model(model, tokenizer, X_test, y_test):
    predictions = []

    for text in X_test:
        inputs = tokenizer(text, return_tensors="pt", truncation=True, padding
    =True)

        with torch.no_grad():
            outputs = model(**inputs)
            pred = torch.argmax(outputs.logits, dim=-1)
            predictions.append(pred.item())

    # Calculate metrics
    accuracy = accuracy_score(y_test, predictions)
    precision, recall, f1, _ = precision_recall_fscore_support(
        y_test, predictions, average='weighted'
    )

    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")

    # Confusion matrix
    cm = confusion_matrix(y_test, predictions)
    print("\nConfusion Matrix:")
    print(cm)

    return predictions
```

Cross-Validation for Robust Results

```
from sklearn.model_selection import StratifiedKFold
import numpy as np

def cross_validate_bert(texts, labels, n_splits=5):
    skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)
    scores = []

    for fold, (train_idx, val_idx) in enumerate(skf.split(texts, labels)):
        print(f"Fold {fold + 1}/{n_splits}")

        X_train = [texts[i] for i in train_idx]
        y_train = [labels[i] for i in train_idx]
        X_val = [texts[i] for i in val_idx]
        y_val = [labels[i] for i in val_idx]

        # Train and evaluate (simplified)
        # In practice, you would fine-tune here
        classifier = pipeline("sentiment-analysis")

        correct = 0
        for text, true_label in zip(X_val, y_val):
            pred = classifier(text)[0]
            pred_label = 1 if pred['label'] == 'POSITIVE' else 0
            if pred_label == true_label:
                correct += 1

        accuracy = correct / len(X_val)
        scores.append(accuracy)

    print(f"\nMean CV Score: {np.mean(scores):.4f} (+/- {np.std(scores):.4f})"
   )
    return scores
```

1.1.7 Practical Applications

Batch Processing for Production

```
def batch_sentiment_analysis(texts, batch_size=32):
    classifier = pipeline("sentiment-analysis", device=0 if torch.cuda.
   is_available() else -1)

    results = []
    for i in range(0, len(texts), batch_size):
        batch = texts[i:i+batch_size]
        batch_results = classifier(batch)
        results.extend(batch_results)

    return results

# Process large dataset efficiently
reviews = ["Review 1...", "Review 2...", ...]  # Thousands of reviews
```

```
sentiments = batch_sentiment_analysis(reviews)
```

Confidence Thresholding

```
def analyze_with_confidence(texts, confidence_threshold=0.8):
    classifier = pipeline("sentiment-analysis")

    high_confidence = []
    needs_review = []

    for text in texts:
        result = classifier(text)[0]

        if result['score'] >= confidence_threshold:
            high_confidence.append({
                'text': text,
                'sentiment': result['label'],
                'confidence': result['score']
            })
        else:
            needs_review.append({
                'text': text,
                'sentiment': result['label'],
                'confidence': result['score']
            })

    print(f"High confidence: {len(high_confidence)}")
    print(f"Needs human review: {len(needs_review)}")

    return high_confidence, needs_review
```

1.1.8 Best Practices

1. Model Selection

- Start with pre-trained models
- Choose domain-specific models when available
- Consider model size vs. performance tradeoff

2. Data Preprocessing

```
def preprocess_for_bert(text):
    # BERT handles most preprocessing, but some cleaning helps
    text = text.lower()
    text = re.sub(r'http\S+', '', text)  # Remove URLs
    text = re.sub(r'@\w+', '', text)  # Remove mentions
    text = re.sub(r'#\w+', '', text)  # Remove hashtags
    text = re.sub(r'\s+', ' ', text)  # Normalize whitespace
    return text.strip()
```

3. Handling Long Texts

```
def analyze_long_text(text, max_length=512):
    # BERT has max length of 512 tokens
    # For longer texts, use sliding window

    tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
    tokens = tokenizer.tokenize(text)

    if len(tokens) <= max_length:
        return classifier(text)
    # Split into chunks with overlap
    chunk_size = 400
    overlap = 100
    chunks = []

    for i in range(0, len(tokens), chunk_size - overlap):
        chunk = tokens[i:i + chunk_size]
        chunk_text = tokenizer.convert_tokens_to_string(chunk)
        chunks.append(chunk_text)

    # Analyze each chunk
    results = [classifier(chunk) for chunk in chunks]

    # Aggregate results (average sentiment scores)
    return aggregate_sentiments(results)
```

1.1.9 Common Challenges and Solutions

Challenge	Solution
Slow inference	Use DistilBERT or batch processing
Out of memory	Reduce batch size or use gradient accumulation
Domain mismatch	Fine-tune on domain-specific data
Multilingual text	Use multilingual BERT variants
Real-time requirements	Cache predictions or use lighter models

1.1.10 Practice Exercise

Create a sentiment analysis system for app reviews:

```
# Task: Analyze app reviews with BERT
def analyze_app_reviews(reviews_file):
    """
    1. Load reviews from CSV
    2. Preprocess text
    3. Analyze sentiment with BERT
    4. Identify common positive/negative themes
    5. Generate insights report
    """

    # Your implementation here
    pass

# Expected output:
# - Overall sentiment distribution
# - Top positive aspects
# - Top negative aspects
# - Reviews needing attention (very negative)
# - Suggested improvements based on sentiment
```

1.1.11 Resources for Further Learning

- Hugging Face Course: https://huggingface.co/course
- BERT Paper: “Attention is All You Need”
- Fine-tuning Tutorial: https://huggingface.co/docs/transformers/training
- Model Hub: https://huggingface.co/models

1.1.12 Assignment

Build a domain-specific sentiment analyzer: 1. Choose a specific domain (e.g., restaurant reviews, product reviews) 2. Collect 100-200 labeled examples 3. Fine-tune a BERT model 4. Evaluate using cross-validation 5. Deploy as a simple API

Deliverables: - Python notebook with implementation - Performance metrics report - API endpoint documentation

---

Advanced Tip: Combine BERT sentiment with rule-based post-processing for domain-specific adjustments!