NLP with TensorFlow: Sentiment Analysis

End-to-end NLP pipeline untuk IMDB sentiment classification menggunakan LSTM + word embeddings. Chapter ini mencakup text preprocessing komprehensif dan production NLP architectures.[1]

Text Exploration dan Preprocessing Pipeline

IMDB Dataset Analysis:

```
def analyze_text(corpus):
    lengths = [len(text.split()) for text in corpus]
    print(f"Avg length: {np.mean(lengths):.1f}")
    print(f"95th percentile: {np.percentile(lengths, 95):.1f}")

    # Vocabulary analysis
    vectorize = TextVectorization(max_tokens=10000)
    vectorize.adapt(corpus)
    print(f"Vocab size: {len(vectorize.get_vocabulary())}")
```

Production Text Pipeline:

```
text_vectorizer = TextVectorization(
    max_tokens=10000,
    output_sequence_length=300,  # 95th percentile
    standardize='lower_and_strip_punctuation',
    output_mode='int'
)
text_vectorizer.adapt(train_texts)
```

```
def text_pipeline(texts, labels):
    dataset = tf.data.Dataset.from_tensor_slices((texts, labels))
    dataset = dataset.map(lambda x, y: (text_vectorizer(x), y))
    return dataset.cache().shuffle(1000).batch(64).prefetch(AUTOTUNE)
```

LSTM-based Sentiment Classifier

Bidirectional LSTM Architecture:

```
def create_sentiment_model(vocab_size=10000, embedding_dim=128):
    inputs = tf.keras.Input(shape=(300,), dtype='int32')

    # Embedding layer
    x = Embedding(vocab_size, embedding_dim)(inputs)

    # Bidirectional LSTM
    x = Bidirectional(LSTM(64, return_sequences=True))(x)
    x = Bidirectional(LSTM(32))(x)

    # Classification head
    x = Dropout(0.5)(x)
    outputs = Dense(1, activation='sigmoid')(x)

    model = Model(inputs, outputs)
    model.compile(
        optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy', 'precision', 'recall']
    )
    return model
```

Word Embeddings Integration   GloVe

Pretrained Embeddings Loading:

```
def load_glove_embeddings(path, vocab, embedding_dim=100):
    embeddings_index = {}
    with open(path) as f:
        for line in f:
            word, *vector = line.split()
            embeddings_index[word] = np.array(vector, dtype='float32')

    embedding_matrix = np.zeros((len(vocab), embedding_dim))
    for i, word in enumerate(vocab):
        embedding_matrix[i] = embeddings_index.get(word, np.random.normal(0, 0.1, embeddi

    return embedding_matrix

# Integrate dengan Keras Embedding layer
embedding_layer = Embedding(vocab_size, embedding_dim,
                          weights=[embedding_matrix], trainable=False)
```

Model Evaluation dan Interpretability

Attention Weights Visualization untuk explainable predictions:

```
def extract_attention_weights(model, text):
    # Custom attention layer untuk LSTM
    attention_model = Model(inputs=model.input,
                          outputs=[model.layers[-3].output, model.output])
    return attention_model.predict(text)
```

Kesimpulan

Bidirectional LSTM + pretrained embeddings mencapai SOTA performance untuk sentiment analysis dengan pipeline production-ready.[1]

```
def split_input_target(sequence):
    input_text = sequence[:-1]
    target_text = sequence[1:]
    return input_text, target_text

def create_sequences(text, seq_length=100):
    total_chars = len(text)
    dataset = tf.data.Dataset.from_tensor_slices(text)
    sequences = dataset.batch(seq_length+1, drop_remainder=True)
    return sequences.map(split_input_target).shuffle(10000).batch(64)
```