Model Optimization and Tuning Phase Template

Date	27 October 2024
Team ID	739894
Project Title	Toxic Comment Classification for Social Media using NLP
Maximum Marks	10 Marks

Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining neural network models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency. "During the optimization and tuning phase for toxic comment classification on social media, techniques such as hyperparameter tuning, model architecture refinement, cross-validation, and evaluation on balanced datasets are employed to improve accuracy, precision, recall, and overall model robustness."

Model	Tuned Hyperparameters
Model 1	(code below)

```
#Fixed: Use train_test instead of train_text
train_test = train_df['comment_text'] # This line was correct
test_train = test_df['comment_text'] # This line was correct
all_text = pd.concat([train_test, test_train]) # Changed train_text to train_test and test_text to te
```

```
[ ] #vectorize the data
    #import and instantiate CountVectorizer
    from sklearn.feature_extraction.text import CountVectorizer
    word_vect = CountVectorizer(
                                strip_accents='unicode',
                                analyzer='word',
                                token_pattern=r'\w{1,}',
                                stop_words='english',
                                ngram_range=(1, 1)
                               )
```

```
# learn the vocabulary in the training data, then use it to create a document-term matrix
word_vect.fit(all_text)
```

	CountVectorizer
	CountVectorizer(stop_words='english', strip_accents='unicode', token_pattern='\\w{1,}')
	
	`from sklearn.model_selection import train_test_split`
	
	`# Assuming 'all_text' is your complete dataset of text documents`
	`train_text, test_text = train_test_split(all_text, test_size=0.2, random_state=42)`
	
	`#transorm the data using the earlier fitted vocabulary, into a doucument-term matrxi`
	`train_features = word_vect.transform(train_text)`
	`test_features = word_vect.transform(test_text)`
	
	`#saving word vectorizer vocab as pkl file to be loaded afterwards`
	`pickle.dump(word_vect.vocabulary_,open('word_feats.pkl','wb'))`

…	`# Clean the input comment`
	`comment = "the comment is toxic"`
	`cleaned_comment = clean_text(comment)`
	
	`# Transform the cleaned comment using the same vectorizer used during trai`
	`comment_features = word_vect.transform([cleaned_comment])  # Make sure to p`
	
	`# Load the models and get predictions for each label`
	`predictions = {}`
	`for label in cols_target:`
	`    # Load the pre-trained model for each label`
	`    model = pickle.load(open(f'{label}_model.sav', 'rb'))`
	
	`    # Get the probability for the positive class (index 1)`
	`    prob = model.predict_proba(comment_features)[:, 1]`
	
	`    # Store the probability in the predictions dictionary`
	`    predictions[label] = prob[0]`
	
	`# Print the predicted probabilities for each target`
	`for label, prob in predictions.items():`
	`    print(f"{label}: {prob:.4f}")`

```python
#specify feature and  target columns
cols_target = [col for col in train_df.columns if 'target_' in col] #adjust if targets follow a specific naming pattern
# Exclude the 'id' column from features
train_features = train_df.drop(columns=cols_target + ['id']) # Add 'id' to dropped columns
#instantiate a dictionary to store models
mapper = {}
#loop over each target column for  Binary Relevance approach
for label in cols_target:
# instantiate the Logistic Regression model
  logreg = LogisticRegression(C=12.0, max_iter=1000)  # increase max_iter if convergence issues arise
  #preapre filename for saving the model
  filename = f"{label}_model.sav"
  print(f"..... processing  {label}")
  #define the target for  the  current label
  y = train_df[label]
#train the model using train_features and target y
  logreg.fit(train_features, y)
#save the trained model
  with open(filename , 'wb') as model_file:
    pickle.dump(logreg, model_file)
#compute the training accuracy
  y_pred_x = logreg.predict(train_features)
  print(f"Training accuarcy for {label}: {accuracy_score(y, y_pred_x)}") # Corrected accuracy_score
#predict probabilities  on test data for the  current label
  test_y_prob = logreg.predict_proba(test_features)[:, 1]
  submission_binary[label] = test_y_prob
```

```python
# Clean the input comment
comment = " bad"
cleaned_comment = clean_text(comment)

# Transform the cleaned comment using the same vectorizer used during training
comment_features = word_vect.transform([cleaned_comment])  # Make sure to pass a list

# Load the models and get predictions for each label
predictions = {}
for label in cols_target:
    # Load the pre-trained model for each label
    model = pickle.load(open(f'{label}_model.sav', 'rb'))

    # Get the probability for the positive class (index 1)
    prob = model.predict_proba(comment_features)[:, 1]
```

<table>
<tr><td></td><td><pre>
    # Store the probability in the predictions dictionary
    predictions[label] = prob[0]

# Print the predicted probabilities for each target
for label, prob in predictions.items():
    print(f"{label}: {prob:.4f}")
✓ 0.0s

toxic: 0.0739
severe_toxic: 0.0189
threat: 0.0023
obscene: 0.0438
insult: 0.0578
identity_hate: 0.0220
</pre></td></tr>
</table>

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
	"For the final model in the optimization and tuning phase of toxic comment classification for social media, we utilized logistic regression classifiers for each target label, trained on features generated by a Count Vectorizer, and fine-tuned using hyperparameter optimization techniques to maximize predictive performance across all categories, the final model employed is logistic regression, optimized using the Count Vectorizer for feature extraction and trained to accurately predict the likelihood of a comment being toxic across various labels.