In [1]:
```
import os
import torch
from torch import nn
from torch.utils.data import DataLoader, Dataset
from transformers import BertTokenizer, BertModel, AdamW, get_linear_schedule_
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_m
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import time
import numpy as np
```

In [2]:
```
# Step 2: Import the IMDB data set and preprocess it
def load_data(data_file):
    # Read the Excel file, explicitly specifying the engine
    df = pd.read_excel(data_file, engine='openpyxl')  # Use the 'openpyxl' eng
    texts = df['Complaint'].tolist()
    labels = df['Category Level 1'].tolist()
    return texts, labels

# Load the data
documents_path = os.path.expanduser('~\\Documents')
os.chdir(documents_path)
data_file = 'final_data.xlsx'
texts, labels = load_data(data_file)

# Reduce dataset size to 10% for easier computation
subset_ratio = 1
subset_size = int(len(texts) * subset_ratio)
texts = texts[:subset_size]
labels = labels[:subset_size]

# Print dataset size
print(f"Number of samples in the dataset: {len(texts)}")
```

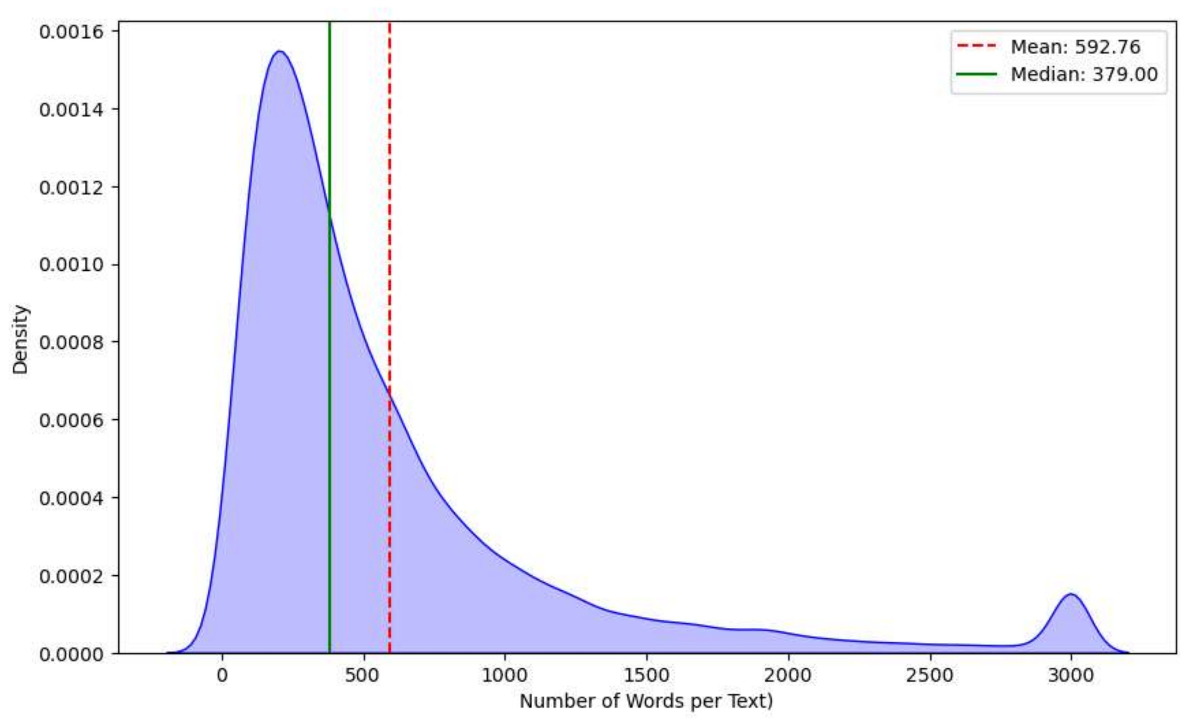
```
Number of samples in the dataset: 73106
```

In [3]:
```python
# Step 3: Create a custom dataset class for text classification
class TextClassificationDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_length):
        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 = self.texts[idx]
        label = self.labels[idx]
        # Ensure text is a string before tokenization
        if not isinstance(text, str):
            text = str(text)  # Convert to string if it's not

        encoding = self.tokenizer(text, return_tensors='pt', max_length=self.m
        return {'input_ids': encoding['input_ids'].flatten(),
                'attention_mask': encoding['attention_mask'].flatten(),
                'labels': torch.tensor(label, dtype=torch.long)}

# Step 4: Build our custom BERT classifier
class BERTClassifier(nn.Module):
    def __init__(self, bert_model_name, num_classes):
        super(BERTClassifier, self).__init__()
        self.bert = BertModel.from_pretrained(bert_model_name)
        self.dropout = nn.Dropout(0.1)
        self.fc = nn.Linear(self.bert.config.hidden_size, num_classes)

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask
        pooled_output = outputs.pooler_output
        x = self.dropout(pooled_output)
        logits = self.fc(x)
        return logits

# Step 5: Define training function
def train(model, data_loader, optimizer, scheduler, device):
    model.train()
    for batch in tqdm(data_loader, desc="Training"):
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        loss = nn.CrossEntropyLoss()(outputs, labels)
        loss.backward()
        optimizer.step()
        scheduler.step()

# Step 6: Build our evaluation method
def evaluate(model, data_loader, device):
    model.eval()
    predictions = []
    actual_labels = []
    with torch.no_grad():
        for batch in data_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)
            outputs = model(input_ids=input_ids, attention_mask=attention_mask
            _, preds = torch.max(outputs, dim=1)
            predictions.extend(preds.cpu().tolist())
            actual_labels.extend(labels.cpu().tolist())
    accuracy = accuracy_score(actual_labels, predictions)
    report = classification_report(actual_labels, predictions)
    return accuracy, report, actual_labels, predictions
```

In [4]:

```python
from scipy import stats
# Calculate the length of each text
lengths = [len(comp) for comp in texts]

# Print mean and median for reference
mean_len = np.mean(lengths)
median_len = np.median(lengths)
mode_len = stats.mode(lengths)
print('Mean length: ', mean_len)
print('Median length: ', median_len)
# Plot the density of text lengths
plt.figure(figsize=(10, 6))
sns.kdeplot(lengths, shade=True, color='blue')

# Add vertical lines for mean and median
plt.axvline(mean_len, color='red', linestyle='--', label=f'Mean: {mean_len:.2f
plt.axvline(median_len, color='green', linestyle='-', label=f'Median: {median_

# Labels and title
plt.xlabel('Number of Words per Text)')
plt.ylabel('Density')
plt.legend()

# Show the plot
plt.show()
```

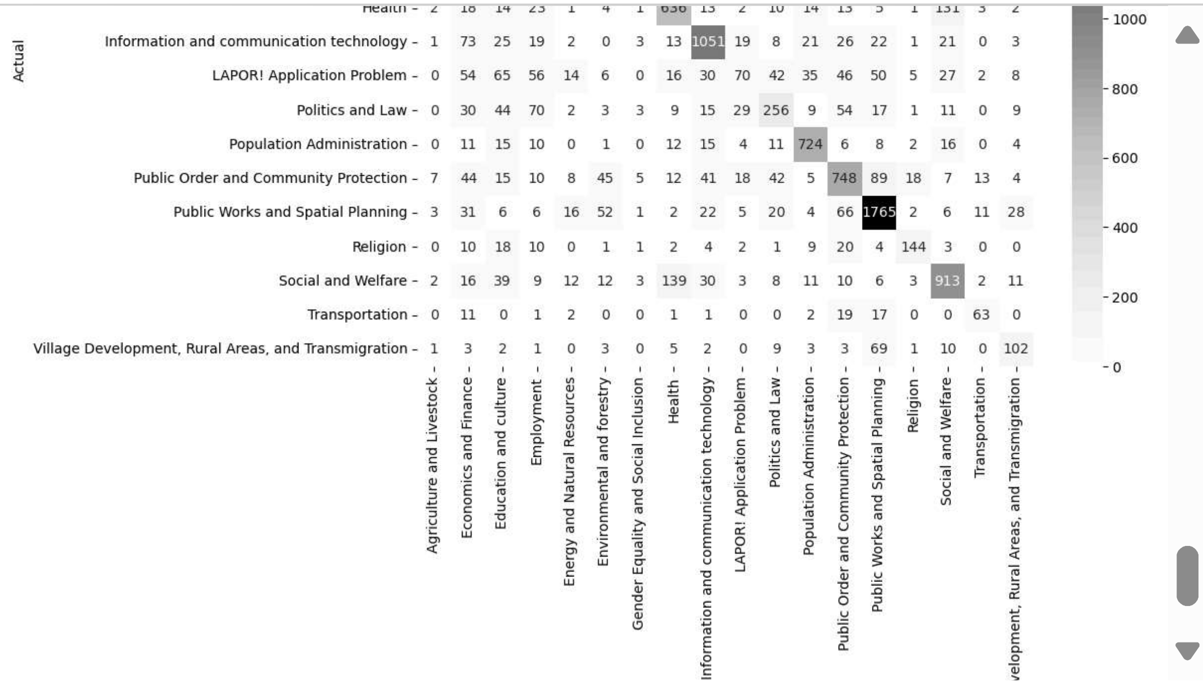
```
Mean length:  592.7580773123957
Median length:  379.0

C:\Users\asus\AppData\Local\Temp\ipykernel_1328\3722292791.py:8: FutureWarnin
g: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default be
havior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0,
this behavior will change: the default value of `keepdims` will become False,
the `axis` over which the statistic is taken will be eliminated, and the valu
e None will no longer be accepted. Set `keepdims` to True or False to avoid t
his warning.
  mode_len = stats.mode(lengths)
C:\Users\asus\AppData\Local\Temp\ipykernel_1328\3722292791.py:13: FutureWarni
ng:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(lengths, shade=True, color='blue')
```



In [5]:

```
# Step 8: Define our model’s parameters
# Set up parameters
bert_model_name = 'indobenchmark/indobert-base-p1'
num_classes = 18
max_length = 300
batch_size = 8
num_epochs = 3
learning_rate = 5e-5

# Step 9: Loading and splitting the data.
train_texts, val_texts, train_labels, val_labels = train_test_split(texts, lab

# Step 10: Initialize tokenizer, dataset, and data loader
# Encode labels BEFORE creating datasets
le = LabelEncoder()
train_labels_encoded = le.fit_transform(train_labels)
val_labels_encoded = le.transform(val_labels)

tokenizer = BertTokenizer.from_pretrained(bert_model_name)
train_dataset = TextClassificationDataset(train_texts, train_labels_encoded, t
val_dataset = TextClassificationDataset(val_texts, val_labels_encoded, tokeniz
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=Tr
val_dataloader = DataLoader(val_dataset, batch_size=batch_size)

# Step 11: Set up the device and model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = BERTClassifier(bert_model_name, num_classes=num_classes).to(device)

# Step 12: Set up optimizer and learning rate scheduler
optimizer = AdamW(model.parameters(), lr=learning_rate)
total_steps = len(train_dataloader) * num_epochs
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0, num

# Training Loop
for epoch in range(num_epochs):
    start_time = time.time()
    print(f"Epoch {epoch + 1}/{num_epochs}")
    train(model, train_dataloader, optimizer, scheduler, device)
    accuracy, report, actual_labels, predictions = evaluate(model, val_dataloa
    end_time = time.time()
    epoch_time = end_time - start_time
    print(f"Validation Accuracy: {accuracy:.4f}")
    print(report)
    print(f"Epoch {epoch + 1} processing time: {epoch_time:.2f} seconds")

print(f"Accuracy: {accuracy:.4f}")

from sklearn.metrics import classification_report

# Assuming you have 'actual_labels' and 'predictions' from the previous run
# Convert numeric labels back to category names
actual_labels_names = le.inverse_transform(actual_labels)
predictions_names = le.inverse_transform(predictions)

# Generate and print the new classification report
new_report = classification_report(actual_labels_names, predictions_names)
print(new_report)

# Calculate the confusion matrix
cm = confusion_matrix(actual_labels, predictions)

# Plot the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Greys", xticklabels=le.classes_, yt
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix for BERT")
plt.show()

# End time
end_time = time.time()
runtime = end_time - start_time
```

```python
print(f"runtime: {runtime:.2f} seconds")
```



In [6]:
```python
# Save the model
torch.save(model.state_dict(), "bert_classifierall_1408.pth")
```

In [ ]: