新闻类别分类

211250109 赵政杰

2023 年 11 月 30 日

1　数据集描述

通过 analysis.py 对数据集进行分析，结果如下：

Number of Samples: 208908

Number of Features: 6

同时分别生成了类别分布的柱状图、饼状图

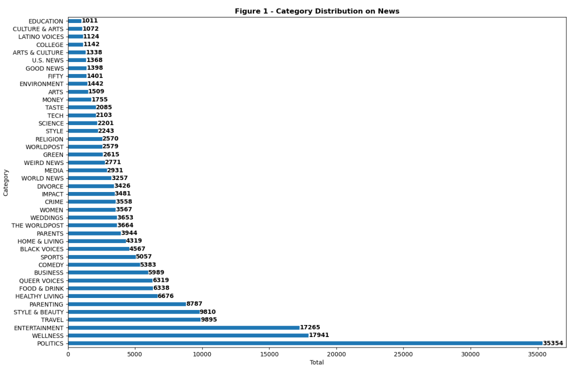

图 1: 柱状图

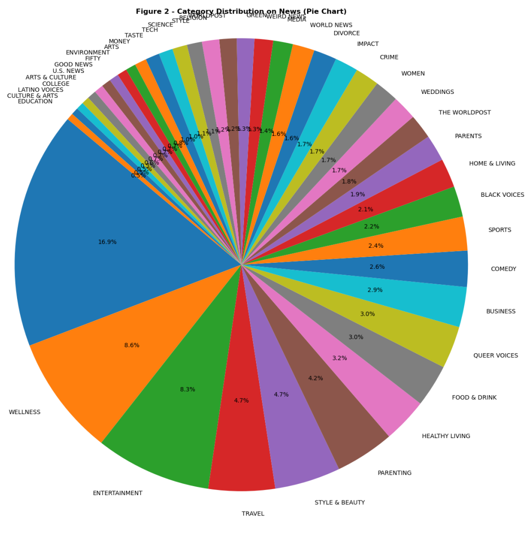

图 2: 饼状图

生成 short_description 列中词语的词云图，如下

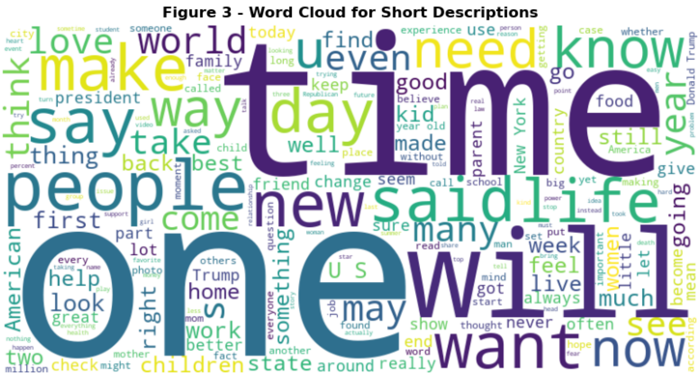

图 3: 词云图

2 数据处理

使用 pandas 库读取 json 文件，并将其存储在 Pandas DataFrame 中。

将**作者、新闻标题、link 和简短描述**合并为一个文本列，并将类别作为标签列构建新的 DataFrame。

定义数据集类 TextClassificationDataset，用来将数据集转化为相应的张量

实现代码如下：

```python
# TextClassificationDataset类定义
class TextClassificationDataset(Dataset):

    def __init__(self,
                 texts: List[str],
                 labels: List[str] = None,
                 label_dict: Mapping[str, int] = None,
                 max_seq_length: int = 512,
                 model_name: str = 'distilbert-base-uncased'):

        self.texts = texts
        self.labels = labels
        self.label_dict = label_dict
        self.max_seq_length = max_seq_length

        if self.label_dict is None and labels is not None:
            self.label_dict = dict(zip(sorted(set(labels)),
                                       range(len(set(labels)))))

        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```python

        self.sep_vid = self.tokenizer.vocab["[SEP]"]
        self.cls_vid = self.tokenizer.vocab["[CLS]"]
        self.pad_vid = self.tokenizer.vocab["[PAD]"]

    def __len__(self):

        return len(self.texts)

    def __getitem__(self, index) -> Mapping[str, torch.Tensor]:

        x = self.texts[index]
        x_encoded = self.tokenizer.encode(
            x,
            add_special_tokens=True,
            max_length=self.max_seq_length,
            return_tensors="pt",
            truncation=True  # 明确启用截断
        ).squeeze(0)

        true_seq_length = x_encoded.size(0)
        pad_size = self.max_seq_length - true_seq_length
        pad_ids = torch.Tensor([self.pad_vid] * pad_size).long()
        x_tensor = torch.cat((x_encoded, pad_ids))

        mask = torch.ones_like(x_encoded, dtype=torch.int8)
        mask_pad = torch.zeros_like(pad_ids, dtype=torch.int8)
        mask = torch.cat((mask, mask_pad))

        output_dict = {
            "features": x_tensor,
            'attention_mask': mask
        }

        if self.labels is not None:
            y = self.labels[index]
            y_encoded = torch.Tensor(
                [self.label_dict.get(y, -1)]
            ).long().squeeze(0)
            output_dict["targets"] = y_encoded

        return output_dict


data_path = "./data"
data = pd.read_json(data_path+"/News_Category.json", lines=True)
text = pd.DataFrame({
    "text" : data.authors+" "+data.headline+" "+data.link+" "+data.short_description,
    "label" : data.category
})
train_dataset = TextClassificationDataset(
    texts=train['text'].values.tolist(),
```

```
    labels=train['label'].values.tolist(),
    label_dict=label_dict,
    max_seq_length=MAX_SEQ_LENGTH,
    model_name=MODEL_NAME
)
valid_dataset = TextClassificationDataset(
    texts=val['text'].values.tolist(),
    labels=val['label'].values.tolist(),
    label_dict=label_dict,
    max_seq_length=MAX_SEQ_LENGTH,
    model_name=MODEL_NAME
)
```

3 分类模型选择及设计、训练、验证、测试

实现在 train.py 文件、test.py 文件和 some_classes.py 文件中

3.1 模型选择与设计

选择使用 DistilBert 模型进行分类。DistilBert 是一种小型、快速和轻量级的 Transformer 模型，通过知识蒸馏 BERT 基础进行训练。

用于序列分类的 DistilBERT 模型定义如下：

```
    class DistilBertForSequenceClassification(nn.Module):

    def __init__(self, pretrained_model_name: str, num_classes: int = None):

        super().__init__()

        config = AutoConfig.from_pretrained(
            pretrained_model_name, num_labels=num_classes)

        self.distilbert = AutoModel.from_pretrained(pretrained_model_name,
                                                    config=config)
        self.pre_classifier = nn.Linear(config.dim, config.dim)
        self.classifier = nn.Linear(config.dim, num_classes)
        self.dropout = nn.Dropout(config.seq_classif_dropout)

    def forward(self, features, attention_mask=None, head_mask=None):

        outputs = self.distilbert(input_ids=features, attention_mask=attention_mask,
            head_mask=head_mask)

        hidden_state = outputs[0]
        pooled_output = hidden_state[:, 0]
        pooled_output = self.pre_classifier(pooled_output)
        pooled_output = nn.ReLU()(pooled_output)
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)

        return logits
```

```
# 加载模型
model = DistilBertForSequenceClassification(pretrained_model_name=MODEL_NAME,
                                            num_classes=NUM_CLASSES)
```

3.2 训练与验证

将数据集划分为 80% 的训练集和 20% 的验证集：

```
train, val = train_test_split(text, test_size=0.20, random_state=SEED)
```

训练参数设置：

```
MODEL_NAME = "distilbert-base-uncased"
NUM_EPOCHS = 3
BATCH_SIZE = 80
MAX_SEQ_LENGTH = 256
LEARN_RATE = 5e-5
SEED = 42
LOG_DIR = 'results'
```

训练与验证过程如代码所示：

```
model = DistilBertForSequenceClassification(pretrained_model_name=MODEL_NAME,
                                            num_classes=NUM_CLASSES)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=LEARN_RATE)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
for epoch in range(NUM_EPOCHS):
    # Training loop
    model.train()
    total_loss = 0

    for batch_idx, batch in enumerate(train_val_loaders['train']):
        optimizer.zero_grad()

        features = batch['features'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        targets = batch['targets'].to(device)

        outputs = model(features, attention_mask=attention_mask)
        loss = criterion(outputs, targets)
        total_loss += loss.item()

        loss.backward()
        optimizer.step()

        if batch_idx % 10 == 0:
        print(f"Epoch [{epoch + 1}/{NUM_EPOCHS}] "
              f"Batch [{batch_idx + 1}/{len(train_val_loaders['train'])}] "
              f"Loss: {loss.item()}")
```

```python
        logging.info(f"Epoch [{epoch + 1}/{NUM_EPOCHS}] "
                     f"Batch [{batch_idx + 1}/{len(train_val_loaders['train'])}] "
                     f"Loss: {loss.item()}")

    average_loss = total_loss / len(train_val_loaders['train'])
    print(f"Epoch [{epoch + 1}/{NUM_EPOCHS}] "
          f"Average training loss: {average_loss}")
    logging.info(f"Epoch [{epoch + 1}/{NUM_EPOCHS}] "
                 f"Average training loss: {average_loss}")

    # Validation loop
    model.eval()
    val_loss = 0
    predicted_labels = []
    true_labels = []

    with torch.no_grad():
        for val_batch in train_val_loaders['valid']:
            features = val_batch['features'].to(device)
            attention_mask = val_batch['attention_mask'].to(device)
            targets = val_batch['targets'].to(device)

            outputs = model(features, attention_mask=attention_mask)
            loss = criterion(outputs, targets)
            val_loss += loss.item()

            _, predicted = torch.max(outputs, 1)

            predicted_labels.extend(predicted.cpu().numpy())  # 预测的标签
            true_labels.extend(targets.cpu().numpy())  # 真实的标签

    average_val_loss = val_loss / len(train_val_loaders['valid'])
    acc = accuracy_score(true_labels, predicted_labels)  # 计算准确率
    precision = precision_score(true_labels, predicted_labels, average='macro')  # 计算精确度
    f1 = f1_score(true_labels, predicted_labels, average='macro')  # 计算F1 Score
    print(f"Epoch [{epoch + 1}/{NUM_EPOCHS}] "
          f"Test Accuracy: {acc}, Test Precision: {precision}, Test F1 Score: {f1}")
    logging.info(f"Epoch [{epoch + 1}/{NUM_EPOCHS}] "
                 f"Test Accuracy: {acc}, Test Precision: {precision}, Test F1 Score: {f1}")

    # 根据验证损失调整学习率
    scheduler.step(average_val_loss)
    # 保存训练好的模型
    torch.save(model.state_dict(), f'results1/trained_model_epoch_{epoch + 1}.pth')
```

3.3 测试

加载训练好的模型文件，在数据集上进行测试，实现如下：

```python
# 加载训练好的模型
```

```python
loaded_model = DistilBertForSequenceClassification(pretrained_model_name=MODEL_NAME,
                                                   num_classes=NUM_CLASSES)
loaded_model.load_state_dict(torch.load('results/trained_model_epoch_2.pth')) # 替换成你想要加载的模型文件名
# 设置模型为评估模式

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
loaded_model.to(device)

def cal_acc_test():
    test_loader = DataLoader(test_dataset, batch_size=100, shuffle=False)
    predicted_labels = []
    true_labels = []

    with torch.no_grad():
        for batch in test_loader:
            features = batch['features'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            targets = batch['targets'].to(device)

            outputs = loaded_model(features, attention_mask=attention_mask)
            _, predicted = torch.max(outputs, 1)

            predicted_labels.extend(predicted.cpu().numpy())  # 预测的标签
            true_labels.extend(targets.cpu().numpy())  # 真实的标签

        acc = accuracy_score(true_labels, predicted_labels)  # 计算准确率
        precision = precision_score(true_labels, predicted_labels, zero_division=1,
            average='macro')  # 计算精确度
        f1 = f1_score(true_labels, predicted_labels, average='macro')  # 计算F1 Score
        print(f"Test Accuracy: {acc}, Test Precision: {precision}, Test F1 Score: {f1}")

if __name__ == "__main__":
    print_classifications_test()
```

3.4 训练结果

参见模型链接中的 log 文件