Project NLP

Automated Customers Reviews

Dusan Dokic / Tiago Ferreira / Carlos Rodríguez

Data Science & Machine Learning Bootcamp 25/10/2024

LOADING DATA SET

1) 1429_1.csv
    - + 41.000 entries

1) Datafiniti_[...].csv
    - 5.000 entries
        - 👍 Positive: 3,478 reviews
        - 😐 Neutral: 1,208 reviews
        - 👎 Negative: 314 reviews

1) Datafiniti_[...]_May19.csv
    - 28,332 entries
        - 👍 Positive: 19,897 reviews
        - 😐 Neutral: 5,648 reviews
        - 👎 Negative: 2,787 reviews

BEFORE PREPROCESSING…

1. Create a ‘sentiment’ column: numerical ‘ratings’ ⟹ categorical ‘ratings’

```python
# Convert ratings to sentiment labels
def convert_to_sentiment(score):
    if score in [1, 2, 3]:
        return 'negative'
    elif score == 4:
        return 'neutral'
    else:
        return 'positive'
```

2. Check for missing values

3. Drop useless columns and keep the essential ones for the project:

— reviews.text
— sentiment
— (reviews.rating) optional
— (categories) optional

```python
# Select relevant columns
df = df[['reviews.text', 'reviews.rating']].copy()  # Use .copy() to avoid modifying the original DataFrame

# Drop rows with missing values in either column
df.dropna(subset=['reviews.text', 'reviews.rating'], inplace=True)
```

1.TEXT PREPROCESSING

```python
# Text Cleaning function
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = text.lower()  # Lowercase
    text = re.sub(r'[^a-zA-Z\s]', '', text)  # Remove special characters
    text = word_tokenize(text)  # Tokenize
    text = [lemmatizer.lemmatize(word) for word in text if word not in stop_words]
    return ' '.join(text)

df['cleaned_text'] = df['reviews.text'].apply(clean_text)
```

def clean_text (text):

— Remove special chars
— Remove numbers
— Remove extra white spaces
— Convert to lowercase
— Remove stopwords
— Tokenization
— Lemmatization

Separate function?
def tokenize_and_lematize (text):

2. MODEL BUILDING…

… BUT FIRST

1. Train-test split

FEATURES	LABELS
i order of them and one of the item is bad qua...	Negative
bulk is always the le expensive way to go for ...	Neutral
well they are not duracell but for the price i...	Positive
seem to work a well a name brand battery at a ...	Positive
these battery are very long lasting the price ...	Positive

2. Vectorization: TF-IDF

```
# Vectorize the text using TF-IDF
vectorizer = TfidfVectorizer(max_df=0.7)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

3. Define hyperparameters

```
# Define hyperparameters for each model
nb_params = {'alpha': [0.5, 1.0, 1.5]}
lr_params = {'C': [0.01, 0.1, 1, 10]}
rf_params = {'n_estimators': [100, 200], 'max_depth': [10, 20, None]}
svm_params = {'C': [0.1, 1, 10], 'kernel': ['linear']}
```

4. Initialize the GridSearchCV

```
# Initialize models with Grid Search
nb_model = GridSearchCV(MultinomialNB(), nb_params, cv=5, scoring='f1_weighted')
lr_model = GridSearchCV(LogisticRegression(max_iter=1000), lr_params, cv=5, scoring='f1_weighted')
rf_model = GridSearchCV(RandomForestClassifier(), rf_params, cv=5, scoring='f1_weighted')
svm_model = GridSearchCV(SVC(), svm_params, cv=5, scoring='f1_weighted')
```

2. MODEL BUILDING TRANSFORMER MODEL

ML MODELs

```
# Model Building with Grid Search for Hyperparameter Tuning

# %%
# Define hyperparameters for each model
nb_params = {'alpha': [0.5, 1.0, 1.5]}
lr_params = {'C': [0.01, 0.1, 1, 10]}
rf_params = {'n_estimators': [100, 200], 'max_depth': [10, 20, None]}
svm_params = {'C': [0.1, 1, 10], 'kernel': ['linear']}

# Initialize models with Grid Search
nb_model = GridSearchCV(MultinomialNB(), nb_params, cv=5, scoring='f1_weighted')
lr_model = GridSearchCV(LogisticRegression(max_iter=1000), lr_params, cv=5, scoring='f1_weighted')
rf_model = GridSearchCV(RandomForestClassifier(), rf_params, cv=5, scoring='f1_weighted')
svm_model = GridSearchCV(SVC(), svm_params, cv=5, scoring='f1_weighted')

# Train models
#nb_model.fit(X_train_tfidf, y_train)
#lr_model.fit(X_train_tfidf, y_train)
rf_model.fit(X_train_tfidf, y_train)
#svm_model.fit(X_train_tfidf, y_train)


# Model Evaluation

# %%
# Evaluation function
def evaluate_model(model, X_test, y_test, model_name):
    y_pred = model.predict(X_test)
    print(f"\n---{model_name}---")
    print(f"Best Params: {model.best_params_}")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(f"Precision: {precision_score(y_test, y_pred, average='weighted'):.4f}")
    print(f"Recall: {recall_score(y_test, y_pred, average='weighted'):.4f}")
    print(f"F1-Score: {f1_score(y_test, y_pred, average='weighted'):.4f}")
    print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_pred)}\n")

# Evaluate all models
#evaluate_model(nb_model, X_test_tfidf, y_test, "Naive Bayes")
#evaluate_model(lr_model, X_test_tfidf, y_test, "Logistic Regression")
evaluate_model(rf_model, X_test_tfidf, y_test, "Random Forest")
#evaluate_model(svm_model, X_test_tfidf, y_test, "SVM")
```

TRANSFORMER MODEL

```
#Load Pre-Trained Tokenizer and Model (e.g., BERT)
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
model = AutoModelForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=3)  # 3 for pos, neg, neu


#Tokenize and Encode Text:

def tokenize_data(texts, tokenizer):
    return tokenizer(texts, padding=True, truncation=True, return_tensors='pt')

X_train_tokens = tokenize_data(X_train.tolist(), tokenizer)
X_test_tokens = tokenize_data(X_test.tolist(), tokenizer)

Run Cell | Run Above
# %% [markdown]
# Fine-tuning the Model

Run Cell | Run Above | Debug Cell
# %%
# Step 1: Import necessary libraries
from sklearn.model_selection import train_test_split
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification, Trainer, TrainingArguments
from datasets import Dataset

# Step 2: Map sentiment labels to integers
sentiment_mapping = {'negative': 0, 'neutral': 1, 'positive': 2}

# Assume df['sentiment'] contains 'positive', 'neutral', 'negative'
# Optionally, sample a smaller dataset for faster experimentation
# Use the entire dataset without sampling
X = df['reviews.text']  # Text data
y = df['sentiment']     # Sentiment labels ('positive', 'neutral', 'negative')

# Step 3: Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Map sentiment labels to integers
y_train_mapped = y_train.map(sentiment_mapping).values
y_test_mapped = y_test.map(sentiment_mapping).values
```

3. MODEL EVALUATION (ML models)

```
---Naive Bayes---
Best Params: {'alpha': 0.5}
Accuracy: 0.7528
Precision: 0.7622
Recall: 0.7528
F1-Score: 0.6877
Confusion Matrix:
[[ 142   41  396]
 [   3  156  934]
 [   2   25 3968]]
```

```
---Logistic Regression---
Best Params: {'C': 10}
Accuracy: 0.8100
Precision: 0.7974
Recall: 0.8100
F1-Score: 0.7965
Confusion Matrix:
[[ 368   54  157]
 [  42  476  575]
 [  52  197 3746]]
```

```
---Random Forest---
Best Params: {'max_depth': None, 'n_estimators': 100}
Accuracy: 0.8703
Precision: 0.8778
Recall: 0.8703
F1-Score: 0.8599
Confusion Matrix:
[[ 361   15  203]
 [  10  630  453]
 [  26   28 3941]]
```

```
---SVM---
Best Params: {'C': 10, 'kernel': 'linear'}
Accuracy: 0.8098
Precision: 0.7980
Recall: 0.8098
F1-Score: 0.7959
Confusion Matrix:
[[ 386   49  144]
 [  56  466  571]
 [  99  159 3737]]
```

3. MODEL EVALUATION (DistilBert model pre-finetuning)

```
Evaluation Metrics for DistilBERT Base Model (without fine-tuning):
Accuracy: 0.6944
Precision: 0.5875
Recall: 0.6944
F1-Score: 0.6308

Confusion Matrix:
[[ 2309     0   478]
 [ 1351     0  4297]
 [ 2533     0 17364]]

Classification Report:
              precision    recall  f1-score   support

    negative       0.37      0.83      0.51      2787
     neutral       0.00      0.00      0.00      5648
    positive       0.78      0.87      0.83     19897

    accuracy                           0.69     28332
   macro avg       0.39      0.57      0.45     28332
weighted avg       0.59      0.69      0.63     28332
```

- Used DestilBert from Huggingface
- Bad “out-of-the-Box” results
- Didn’t get any neutral reviews

3. MODEL EVALUATION (RandomForest on Eval Dataset)

```
---Random Forest Model Evaluation---
Accuracy: 0.8440
Precision: 0.8641
Recall: 0.8440
F1-Score: 0.8265
Confusion Matrix:
 [[ 177    6  131]
 [   2  588  618]
 [   3   20 3455]]

Classification Report:
              precision    recall  f1-score   support

    negative       0.97      0.56      0.71       314
     neutral       0.96      0.49      0.65      1208
    positive       0.82      0.99      0.90      3478

    accuracy                           0.84      5000
   macro avg       0.92      0.68      0.75      5000
weighted avg       0.86      0.84      0.83      5000
```

- Evaluated the RF model on a separate Dataset(Datafiniti_[...].csv 5000 rows)

- Seemingly good accuracy

3. MODEL EVALUATION (DistilBert on Eval Dataset)

```
Classification Report:
              precision    recall  f1-score   support

    negative       0.85      0.82      0.84       273
     neutral       0.35      0.25      0.29       211
    positive       0.90      0.94      0.92      1516

    accuracy                           0.85      2000
   macro avg       0.70      0.67      0.68      2000
weighted avg       0.83      0.85      0.84      2000
```

- Evaluated the DistilBert model on a separate Dataset(Datafiniti_[...].csv 5000 rows)

- Accuracy is alright for positive and negative cases but this model really struggles with neutral one

DEMO

Future Work

To improve the model, the following things could be done:

- Balance the dataset out either by removing positive and negative reviews or
- Augment the data to have more neutral reviews
- Look up a better model on Huggingface to use as a base