nlp-final-assignment

December 30, 2023

0.1 Dataset Overview

[1]:
```
import pandas as pd
import string
from nltk.corpus import stopwords
import nltk
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

#used models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.svm import SVC

#Evaluation
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix,␣
 ↪recall_score, precision_score
from sklearn.model_selection import RandomizedSearchCV
```

[2]:
```
data = pd.read_csv("amazon.csv")
```

[3]:
```
data.head()
```

[3]:
```
                                          reviewText  Positive
0  This is a one of the best apps acording to a b…         1
1  This is a pretty good version of the game for …         1
2  this is a really cool game. there are a bunch …         1
3  This is a silly game and can be frustrating, b…         1
4  This is a terrific game on any pad. Hrs of fun…         1
```

[4]:
```
data.describe()
```

[4]:
```
           Positive
count  20000.000000
mean       0.761650
std        0.426085
```

```
min        0.000000
25%        1.000000
50%        1.000000
75%        1.000000
max        1.000000
```

[5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   reviewText  20000 non-null  object
 1   Positive    20000 non-null  int64
dtypes: int64(1), object(1)
memory usage: 312.6+ KB
```

Description of columns

```
1. 'reviewText': textual content of the review
2. 'Positive': binary label (1 for positive, 0 for negative)
```

0.2 Data Preprocessing

[6]:
```
data2 = data.copy()
```

[7]:
```
#Checking Imbalanced Dataset
a = data['Positive'].value_counts()[0]
b= data['Positive'].value_counts().sum()

(a/b) * 100
```

[7]:
```
23.835
```

[8]:
```
punctuations = string.punctuation

def remove_punc(text):
    final_text = []
    for i in text:
        if i not in punctuations:
            final_text.append(i)
    final_text = ''.join(final_text)
    return final_text
```

[9]:
```
stopword = set(stopwords.words('english'))

def remove_st(text):
    text = text.lower()
    final_text = [word for word in text.split() if word not in stopword]
    return final_text
```

[10]:
```
data['reviewText'] = data['reviewText'].apply(remove_punc)
data['reviewText'] = data['reviewText'].apply(remove_st)
```

Lemmatize Text

[11]:
```
lemmatizer = WordNetLemmatizer()

def lemm_text(text_list):
    text = [lemmatizer.lemmatize(word) for word in text_list]
    text = ' '.join(text)
    return text
```

[12]:
```
data['reviewText'] = data['reviewText'].apply(lemm_text)
```

Vectorize Text

[13]:
```
vectorizer = TfidfVectorizer()

x = vectorizer.fit_transform(data['reviewText'])
y = data['Positive']
```

Split Dataset

[14]:
```
X_Train, X_Test, Y_Train, Y_Test = train_test_split(x, y, test_size=.3,␣
  ↪random_state=42)
```

[15]:
```
X_Train.shape
```

[15]:
```
(14000, 22084)
```

0.2.1 Model Selection and Training

[16]:
```
models = [
    LogisticRegression(),
    RandomForestClassifier(),
    MultinomialNB(),
    BernoulliNB(),
    SVC()
]
```

[17]:
```
def train_md(model, X, Y, testX):
    model.fit(X, Y)
    predicted = model.predict(testX)
    model_name = type(model).__name__
    return model_name, predicted
```

[18]:
```
model_info = {}
```

[19]:
```
for model in models:
    name, pred = train_md(model, X_Train, Y_Train, X_Test)
    model_info[name] = {'actual_Y': Y_Test, 'predicted_Y': pred}
```

0.2.2 Formal Evaluation

[20]:
```
def get_score(x, y):
    sc = {
            'accuracy' : accuracy_score(x, y),
            'f1' : f1_score(x, y),
            'recall' : recall_score(x, y),
            'precision' : precision_score(x, y),
            'cm' : confusion_matrix(x, y),
    }
    return sc
```

[21]:
```
score = {}

for model_name in model_info:
    sc = get_score(model_info[model_name]['actual_Y'],␣
  ↪model_info[model_name]['predicted_Y'])
    score[model_name] = sc
```

[22]:
```
for name in score:
    print("-----------*******-----------")
    print(name)
    for i in score[name]:
        print(i, ":", score[name][i])
```

```
-----------*******-----------
LogisticRegression
accuracy : 0.8848333333333334
f1 : 0.9280732799000729
recall : 0.9714534757027675
precision : 0.8884017536867278
cm : [[ 851  560]
 [ 131 4458]]
-----------*******-----------
RandomForestClassifier
accuracy : 0.8625
f1 : 0.9156355455568054
recall : 0.9755938112878623
precision : 0.8626204238921001
```

```
cm : [[ 698  713]
 [ 112 4477]]
-----------*******-----------
MultinomialNB
accuracy : 0.7885
f1 : 0.8782032824647279
recall : 0.9969492264109828
precision : 0.7847341337907375
cm : [[ 156 1255]
 [  14 4575]]
-----------*******-----------
BernoulliNB
accuracy : 0.865
f1 : 0.915625
recall : 0.9577249945521901
precision : 0.8770704450209539
cm : [[ 795  616]
 [ 194 4395]]
-----------*******-----------
SVC
accuracy : 0.8888333333333334
f1 : 0.9303248720359345
recall : 0.97036391370669
precision : 0.8934590690208668
cm : [[ 880  531]
 [ 136 4453]]
```

0.2.3 HyperParameter Tuning

[23]:
```
HP_model = {
    'lr': {'model': LogisticRegression(),
            'params':{
                'penalty': ['l1', 'l2'],
                'C': [1.0, 2.0],
                'solver': ['liblinear'],
                'max_iter': [20, 50, 100],
    }
},
    'rf': {'model': RandomForestClassifier(),
            'params':{
                'n_estimators': [20, 50, 100],
                'criterion': ['gini', 'entropy'],
                'min_samples_leaf': [1, 2],
                'max_features': ['sqrt', 'log2']
    }
},
    'mb': {'model': MultinomialNB(),
            'params':{
                'force_alpha': [True, False],
                'fit_prior': [True, False]
            }
},
    'bb': {'model': BernoulliNB(),
            'params':{
                'alpha': [1.0, 2.0],
                'force_alpha': [True, False]
            }
},
    'svc': {'model': SVC(),
            'params':{
                'C': [1.0, 2.0],
                'kernel': ['linear', 'rbf'],
            }
}
}
```

[24]:
```
HP_res = {}
```

[25]:
```
for mName in HP_model:
    model = HP_model[mName]['model']
    params = HP_model[mName]['params']
    clf = RandomizedSearchCV(model, param_distributions=params, random_state=0)
    clf = clf.fit(X_Train, Y_Train)
    HP_res[mName] = {'model': model, 'score': clf.best_score_, 'Bparams': clf.
  ↪best_estimator_}
```

```
/home/applehx7/anaconda3/lib/python3.11/site-
packages/sklearn/model_selection/_search.py:305: UserWarning: The total space of
parameters 4 is smaller than n_iter=10. Running 4 iterations. For exhaustive
searches, use GridSearchCV.
  warnings.warn(
/home/applehx7/anaconda3/lib/python3.11/site-
packages/sklearn/model_selection/_search.py:305: UserWarning: The total space of
parameters 4 is smaller than n_iter=10. Running 4 iterations. For exhaustive
searches, use GridSearchCV.
  warnings.warn(
/home/applehx7/anaconda3/lib/python3.11/site-
packages/sklearn/model_selection/_search.py:305: UserWarning: The total space of
parameters 4 is smaller than n_iter=10. Running 4 iterations. For exhaustive
searches, use GridSearchCV.
  warnings.warn(
```

[26]:
```
HP_res
```

```
[26]: {'lr': {'model': LogisticRegression(),
        'score': 0.8885714285714286,
        'Bparams': LogisticRegression(C=2.0, solver='liblinear')},
       'rf': {'model': RandomForestClassifier(),
        'score': 0.8654999999999999,
        'Bparams': RandomForestClassifier(criterion='entropy')},
       'mb': {'model': MultinomialNB(),
        'score': 0.8447857142857144,
        'Bparams': MultinomialNB(fit_prior=False, force_alpha=True)},
       'bb': {'model': BernoulliNB(),
        'score': 0.8521428571428571,
        'Bparams': BernoulliNB(force_alpha=True)},
       'svc': {'model': SVC(), 'score': 0.8951428571428572, 'Bparams': SVC(C=2.0)}}
```

0.2.4 Explanation of Chosen Hyperparameters:

1. **Logistic Regression (lr):**
    - penalty: Specifies the norm used in the penalization. l1 and l2 are commonly used penalties for regularization.
    - C: Regularization parameter. It controls the trade-off between fitting the training data well and keeping the model simple.
    - solver: Algorithm used for optimization. 'liblinear' is suitable for small-to-medium-sized datasets.
    - max_iter: Maximum number of iterations taken for the solvers to converge.
2. **Random Forest (rf):**
    - n_estimators: The number of trees in the forest. Different values like [20, 50, 100] are tested to find the optimal number of trees.
    - criterion: Function to measure the quality of a split. 'gini' and 'entropy' are criteria for impurity.
    - min_samples_leaf: Minimum number of samples required to be at a leaf node.
    - max_features: The number of features to consider when looking for the best split.
3. **Multinomial Naive Bayes (mb):**
    - force_alpha: This parameter is not a standard hyperparameter for MultinomialNB in scikit-learn. It might be specific to your implementation or a custom hyperparameter.
    - fit_prior: Boolean parameter specifying whether to learn class prior probabilities or not.
4. **Bernoulli Naive Bayes (bb):**
    - alpha: This parameter is used for Laplace smoothing. Different values like [1.0, 2.0] are tested.
    - force_alpha: Similar to the force_alpha parameter in MultinomialNB, this may be a custom hyperparameter.
5. **Support Vector Classifier (svc):**
    - C: Penalty parameter for the error term. Different values like [1.0, 2.0] are tested.

- kernel: Type of kernel used in the algorithm. ‘linear’ and ‘rbf’ are commonly used kernels.

0.2.5 Reasoning behind the Chosen Hyperparameters:

- **Exploration of Different Aspects:** The chosen hyperparameters cover various aspects of each model, such as regularization, model complexity, kernel choice, and criterion for split.
- **Range Selection:** The ranges provided seem to be a good starting point for exploration. For example, testing different values of C, n_estimators, or different penalties in Logistic Regression and Naive Bayes.
- **Common Defaults:** Some hyperparameters like C, kernel, penalty, and n_estimators are widely used defaults and essential for model behavior control.
- **Model-Specific Parameters:** There are instances of potentially model-specific parameters like force_alpha in Naive Bayes, which might be used for custom implementations or specific requirements.

0.2.6 Comaparative Analysis

[27]:
```
results = {}

for mName in HP_res:
    clf = HP_res[mName]['Bparams']
    model = HP_res[mName]['model']
    y_pred = clf.predict(X_Test)
    accuracy = accuracy_score(Y_Test, y_pred)
    precision = precision_score(Y_Test, y_pred)
    recall = recall_score(Y_Test, y_pred)
    f1 = f1_score(Y_Test, y_pred)
    cm = confusion_matrix(Y_Test, y_pred)

    results[type(model).__name__] = {'Accuracy': accuracy, 'Precision':␣
 ↪precision, 'Recall': recall, 'F1 Score': f1, 'Confusion Matrix': cm}


# Print evaluation metrics for each model
for mName, metrics in results.items():
    print(f"Model: {mName}")
    print(metrics)
```

```
Model: LogisticRegression
{'Accuracy': 0.8918333333333334, 'Precision': 0.9023692810457516, 'Recall':
0.9627369797341468, 'F1 Score': 0.9315761729045862, 'Confusion Matrix': array([[
933,  478],
       [ 171, 4418]])}
Model: RandomForestClassifier
{'Accuracy': 0.8651666666666666, 'Precision': 0.8664210934470725, 'Recall':
```

```
0.9738505120941382, 'F1 Score': 0.9170001025956706, 'Confusion Matrix': array([[
722,  689],
       [ 120, 4469]])}
Model: MultinomialNB
{'Accuracy': 0.855, 'Precision': 0.8609978644923316, 'Recall':
0.9664414905208106, 'F1 Score': 0.9106776180698153, 'Confusion Matrix': array([[
695,  716],
       [ 154, 4435]])}
Model: BernoulliNB
{'Accuracy': 0.865, 'Precision': 0.8770704450209539, 'Recall':
0.9577249945521901, 'F1 Score': 0.915625, 'Confusion Matrix': array([[ 795,
616],
       [ 194, 4395]])}
Model: SVC
{'Accuracy': 0.893, 'Precision': 0.90515294600698, 'Recall': 0.9607757681412072,
'F1 Score': 0.9321353065539112, 'Confusion Matrix': array([[ 949,  462],
       [ 180, 4409]])}
```

0.2.7 Identify the strengths and weaknesses of each model.

1. **Logistic Regression:**
    1. Strengths:
        - Provides probabilities for predictions.
        - Works well with a large number of features.
    2. Weaknesses:
        - Prone to underperform if features are not linearly separable.
        - May not capture complex relationships between features.
2. **Random Forest Classifier:**
    1. Strengths:
        - Handles non-linearity and interactions well.
        - Works with both numerical and categorical data.
        - Less prone to overfitting compared to decision trees.
    2. Weaknesses:
        - Can be computationally expensive for large datasets.
        - Interpretability might be challenging with a large number of trees.
3. **Multinomial Naive Bayes:**
    1. Strengths:
        - Efficient and simple algorithm.
        - Handles high-dimensional data well, often used in text classification.
    2. Weaknesses:
        - Assumes independence between features (features are conditionally independent given the class), which might not hold true in real-world scenarios.
        - Doesn't handle negative values well (as it's specifically designed for features representing word counts).
4. **Bernoulli Naive Bayes:**

   1. Strengths:
      - Suitable for binary/boolean features.
      - Performs well with a small amount of training data.
   2. Weaknesses:
      - Similar to Multinomial Naive Bayes, assumes feature independence.
      - Sensitive to feature distributions; doesn't perform well with continuous or non-binary data.

5. **Support Vector Classifier (SVC):**

   1. S- trengths: Effective in high-dimensional spaces.
      - Versatile due to different kernel options for non-linear classification.
   2. Weaknesses:
      - Computationally expensive with large datasets.
      - Sensitivity to the choice of kernel and regularization parameters.

0.3 Conclusion

The analysis aimed to assess various machine learning models for sentiment analysis using Amazon product reviews. Five models, namely Logistic Regression, Random Forest, Multinomial Naive Bayes, Bernoulli Naive Bayes, and Support Vector Classifier (SVC), were evaluated based on their performance metrics on a test dataset.

The findings revealed diverse performances among the models, each demonstrating distinct strengths and weaknesses. Notably, the Support Vector Classifier (SVC) exhibited slightly superior accuracy and F1 score compared to the other models. However, each model showcased specific characteristics:

- **Logistic Regression** performed consistently well and provided probabilities for predictions, but its performance heavily relied on linear separability of features.
- **Random Forest Classifier** demonstrated robustness against overfitting, handling non-linearity effectively. However, its interpretability and computational cost might be challenges in some scenarios.
- **Naive Bayes** models—Multinomial and Bernoulli—proved efficient and straightforward but relied on the independence assumption among features, which might not hold in all situations.
- **SVC** showcased strong predictive capabilities in high-dimensional spaces, but its computational intensity and sensitivity to kernel selection were noteworthy limitations.

```
[ ]:
```