a2

April 25, 2022

1 COMPSCI 762 Assignment 2 Naive Bayes Classifier

Chase Robertson
crob873

1.1 Motivation

In order to match the benchmark model performance initially, some default parameters had to be modified straightaway. I chose to ignore prior probabilities in the Naive Bayes' probability calculation on suspicion that the training data's category distribution was imbalanced. I also chose to tune the model's smoothing parameter, as that seemed a necessary change to account for words with zero count. After making those two changes, I was able to closely match the benchmark model's performance. Further improvement to prediction scores then needed to be achieved by other means.

1.1.1 Task 1

I chose to first attempt to improve prediction by excluding english stop words from the word frequency vector. I also attempted to include bigrams in the vector, and tried turning off the default inverse document frequency calculation executed by the term frequency utility provided by `sklearn`. I also switched to the Rennie et al. (2003) version of Multinomial Naive Bayes model to try to address the issues illuminated in that paper.

1.1.2 Task 2

I chose to add the `name` and `mean_checkin_time` attributes to the model to try to improve prediction scores. Some establishment names explicitly include class information, so the benefit of including that attribute is obvious. I chose to represent that attribute as a bag of words in the same or similar way as the `review` attribute. I suspected that information about the time spent checking in to the establishment may be helpful as well, as some categories may be associated with longer or shorter checkins. The only other attributes available in the dataset are latitude and longitude, which do not seem very informative since different types of businesses can be mixed in the same area. There could be some association between output class and location due to local tradition or zoning policy, but it would surely require some clever preprocessing for the Naive Bayes model to discover those associations.

1.2 Data Representation and Preprocessing

Each observation is initially represented by a term frequency vector, constructed from the `review` attribute. The default behavior of `sklearn`'s `TfidfTransformer` is to incorporate the inverse document frequency in each term frequency. In Task 1, that default behavior is turned off, so basic term frequency is used. English stop words are removed from the term frequency vector in both Task 1 and 2. Bigrams from the `review` feature are unsuccessfully included in Task 1, but somewhat successfully included in Task 2, with the difference in effect probably due to the additional maximum limit placed on the number of features in Task 2.

1.3 Implementation

Initial implementation focused on boilerplate setup and matching prediction performance with the benchmark level of about 87%. This was achieved by removing prior probabilities from prediction, and tuning the smoothing parameter.

1.3.1 Task 1

Though many changes were attempted in Task 1, the real improvement in prediction came from the combination of removing english stop words and skipping the inverse document frequency calculation. These changes, in concert with the use of Rennie et al. (2003) model, led to a consistent prediction accuracy of almost 90%. I believe the removal of stop words and use of simpler term frequency vectors enabled the more robust `ComplementNB` model to resist bias to the training data.

1.3.2 Task 2

Adding the `name` and `mean_checkin_time` attributes in Task 2 alone led to a small improvement in prediction accuracy. When considering the addition of more term frequency features, it occurred to me that there may be thousands of features already in the model that were not very informative. Adding the `max_features` argument to the Naive Bayes model, in combination with the additional attributes and re-tuning of existing hyperparameters, led to a significant improvement over the benchmark model.

1.4 Evaluation Procedure

All models were evaluated and their hyperparameters selected by 5-fold cross-validation. I decided that cross-validation was necessary to maximize the number of observations available for training, since only a few thousand observations were available in total. Fewer cross-validation folds were used for intermediate exploration, as computation time of exhaustive grid search over many hyperparameters became an issue.

1.5 Validation Results

The validation accuracy of each model is listed below. Again, this was calculated using 5-fold cross-validation to maximize the exposure of each model to the small training set.

- Task 0 (benchmark): **88.5%**
- Task 1 (optimised): **89.4%**
- Task 2 (additional attributes): **91.4%**

2 Code

2.1 Part 0a - Setup

[1]:
```
import os
import numpy as np
import pandas as pd
from pathlib import Path
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import ComplementNB
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler

np.random.seed(12345678)

PATH_ROOT = Path(os.getcwd())
train_file = os.path.join(PATH_ROOT, 'train.csv')
test_file = os.path.join(PATH_ROOT, 'test.csv')
benchmark_file = os.path.join(PATH_ROOT, 'benchmark_predict.csv')
improved_file = os.path.join(PATH_ROOT, 'improved_predict.csv')
plus_attr_file = os.path.join(PATH_ROOT, 'plus_attr_predict.csv')

train = pd.read_csv(train_file)
X_train_review = train['review']
y_train = train['category']
test = pd.read_csv(test_file)
X_test_review = test['review']
X_test_id = test['ID']
```

2.2 Part 0b - Replicate Benchmark Model

[2]:
```
# Replicate Benchmark Model
pipe = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB()),
])

bench_params = {
    'clf__alpha': np.arange(0.2, 0, -0.05),
    'clf__fit_prior': [True, False],
}
```

```
bench_cv = GridSearchCV(pipe, bench_params, n_jobs=-1)
bench_model = bench_cv.fit(X_train_review, y_train)

print("Best validation score: %r" % (bench_model.best_score_))
print("----- Achieved with -------")
for name in sorted(bench_params.keys()):
    print("%s: %r" % (name, bench_model.best_params_[name]))
```

```
Best validation score: 0.885144675049235
----- Achieved with -------
clf__alpha: 0.10000000000000003
clf__fit_prior: False
```

[3]:

```
# write predictions to file
y_test_pred = pd.Series(bench_model.predict(X_test_review),
                        name='category')
submission = pd.concat([X_test_id, y_test_pred], axis=1)
submission.to_csv(benchmark_file, index=False)
```

2.3 Part 1 - Improve based on Review only

[4]:

```
# Improve Benchmark Model with complement model and new params
improved_pipe = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    # IMPROVEMENT use model from Rennie et al. 2003
    ('clf', ComplementNB()),
])

improved_params = {
    # IMPROVEMENT params
    'vect__stop_words': ['english'],
    'vect__ngram_range': [(1,1), (1,2)],
    'tfidf__use_idf': [True, False],
    # BENCHMARK params
    'clf__alpha': [0.1],
    # fit_prior unneccessary - only applies to edge cases in ComplementNB
}

improved_cv = GridSearchCV(improved_pipe, improved_params, cv=5, n_jobs=-1)
improved_model = improved_cv.fit(X_train_review, y_train)

print("Improvement over benchmark: %r" %(improved_model.best_score_ -␣
 ↪bench_model.best_score_))
print("Best validation score: %r" % (improved_model.best_score_))
print("----- Achieved with -------")
for name in sorted(improved_params.keys()):
```

```
    print("%s: %r" % (name, improved_model.best_params_[name]))
```

```
Improvement over benchmark: 0.008695046205120405
Best validation score: 0.8938397212543554
----- Achieved with -------
clf__alpha: 0.1
tfidf__use_idf: False
vect__ngram_range: (1, 1)
vect__stop_words: 'english'
```

[5]:

```
# write predictions to file
y_test_pred = pd.Series(improved_model.predict(X_test_review),
                        name='category')
submission = pd.concat([X_test_id, y_test_pred], axis=1)
submission.to_csv(improved_file, index=False)
```

2.4 Part 2

[6]:

```
# Improve model with additional attributes

# Add 'name' attr and use the same BoW preprocess as 'review'
words_attrs = ['review', 'name']
words_pipe = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
])

# Add mean checkin time without any preprocessing
num_attrs = ['mean_checkin_time']

preprocessor = ColumnTransformer([
    ('review', words_pipe, 'review'),
    ('name', words_pipe, 'name'),
    ('num', 'passthrough', num_attrs),
])

best_pipe = Pipeline([
    ('pre', preprocessor),
    ('clf', ComplementNB()),
])

best_params = {
    # TASK 2 IMPROVEMENT params
    'pre__review__vect__max_features': np.arange(3000, 5000, 500),
    # TASK 1 IMPROVEMENT params
    'pre__review__vect__stop_words': ['english'], # , None],
    'pre__review__vect__ngram_range': [(1,2)], #, (1,1)],
```

```
    #'pre__name__vect__ngram_range': [(1,1), (1,2)],
    #'pre__review__tfidf__use_idf': [False], #, True],
    'pre__name__tfidf__use_idf': [False], #, True],
    # BENCHMARK params
    'clf__alpha': np.arange(0.34, 0.25, -0.02), #np.arange(0.3, 0.22, -0.02),
    'clf__fit_prior': [True]#, False],
}

best_cv = GridSearchCV(best_pipe, best_params, cv=5, n_jobs=-1)

best_model = best_cv.fit(train, y_train)

print("Improvement over benchmark: %r" %(best_model.best_score_ - bench_model.
 ↪best_score_))
print("Improvement over Part 1: %r" %(best_model.best_score_ - improved_model.
 ↪best_score_))
print("Best validation score: %r" % (best_model.best_score_))
print("----- Achieved with -------")
for name in sorted(best_params.keys()):
    print("%s: %r" % (name, best_model.best_params_[name]))
```

```
Improvement over benchmark: 0.028885320405999004
Improvement over Part 1: 0.0201902742008786
Best validation score: 0.914029995455234
----- Achieved with -------
clf__alpha: 0.32
clf__fit_prior: True
pre__name__tfidf__use_idf: False
pre__review__vect__max_features: 3500
pre__review__vect__ngram_range: (1, 2)
pre__review__vect__stop_words: 'english'
```

[7]:
```
# write predictions to file
y_test_pred = pd.Series(best_model.predict(test),
                        name='category')
submission = pd.concat([X_test_id, y_test_pred], axis=1)
submission.to_csv(plus_attr_file, index=False)
```