SAKI SS19 Homework 1

Author: Maximilian Lattka

Program code: https://github.com/nakami/oss-saki-ss19-exercise-1/releases/tag/oss-saki-ss19-exercise-1

Summary

The given data set contains 209 rows of transactions manually annotated with 6 different labels (“finance”, “income”, “leisure”, “living”, “private” and “standardOfLiving”). For the classification task I chose to combine the columns “Buchungstext”, ”Verwendungszweck” and “Beguenstigter/Zahlungspflichtiger” as relevant data whereas the incidences of words per label are used (bag-of-words). If I’d continued without further adjustions, the bag-of-words would contain 423 words/features. However, the data is rather noisy with many common abbreviations and numbers. I would like the classifier to look at certain keywords which strongly indicate a certain category/type. Regarding preprocessing, I chose to omit numbers, because text strings like “00”, “00eur0” or “07folgenr” don’t contribute much to the classification intuitively. My reasoning is that Naive Bayes Classifiers work well for text categorization (the categories being the labels) when used with word frequencies, but not with numerical data. At this point 237 words remain. I then checked the remaining words and developed a suitable list of 16 stop words which are omitted as well. Some are the remainder of account or reference IDs which contained numbers and letters but have been crunched by leaving out the numbers in the previous steps, some are common words which aren’t contributing to the classification either. In the evaluation section I will come back to the relevance of the stop word list and talk about further omitting the occurrences of “euro” (`[evalC]`). Finally, the amount of words or features in the bag-of-words after preprocessing the data could be reduced from 423 to 221 (or to 218 for the feature set for `[evalC]`).

Evaluation

I evaluated the `GaussianNB`-classifier with three different feature sets using the `KFold`-class provided by `sklearn.model_selection`. Section `[evalA]` evaluates the combination with the feature set without numbers and stop words, section `[evalB]` on the other hand the feature set if I was to not do any preprocessing or leaving out words. For section `[evalC]` the feature set appends to the one of section `[evalA]` by further omitting different spellings of “euro” to attack overfitting. For `KFold`, I chose `k=10` and therefore ended up with 10 splits with each having 90% of the data rows as training and 10% of the data rows as testing set. The distribution of the labels is rather balanced (see the jupyter notebook), so the accuracy metric (`accuracy_score` from `sklearn.metrics`) is fine. In terms of actual results one needs to be careful as - like already mentioned - the data set contains very few samples and high scores may occur because of overfitting. For the feature set of 221 words (numbers and stop words omitted), I take the arithmetic mean of the accuracy score is computed and results in `0.9` with the `shuffle`-parameter for `KFold` set to `False` (see `[evalA]` section in jupyter notebook). Enabling the `shuffle`-parameter, I observe the arithmetic mean accuracy score fluctuating between `0.88` and `0.93` (`n=50`). Taking a look at the results of the data set without preprocessing (423 words/features), we end up with a slightly higher mean accuracy score (≅`0.92`), but are theoretically prone to overfitting (see `[evalB]` section in jupyter notebook). Another interesting observation is the resulting score when appending the words “eu”, “eur” and “euro” (see `[evalC]` section in jupyter notebook). This feature set includes with 218 words/features even less than the one shown in the `[evalA]`-section (221), but yields a slight decrease regarding its mean accuracy score by leaving out the features for three spelling for “euro”: less than `0.88`. For this set the arithmetic mean accuracy score fluctuates between `0.88` and `0.92` (`n=50`) with the `shuffle`-parameter set to `True`. Here one may argue towards these features tending to overfit and being legitimately omitted in this feature set or not and therefore are rightful categorization clues for certain labels. Although, the trained `GaussianNB`-classifier in `[evalC]` having a lower score than being trained , I would prefer using it in comparison to `[evalA]` when experiencing new unknown real-world data samples. In my eyes the term “euro” and all its different spellings theoretically may appear in all sorts of transactions. It may be also worthwhile to investigate even further and check where bad predictions were made. For

the worst split (accuracy mean of `0.71`) of `[evalC]` I computed the confusion matrix. Prominent here is that true "leisure"-samples are wrongly predicted as "standardOfLiving" or "living". This points out the problem of the categories not being clearly disjunct and overlap idiomatically. Where exactly were the boundaries when the data set was originally annotated? I also want to stress again that the provided data set contains very few samples for any general machine learning approach, so expanding the data set may yield better results without increasing the mentioned risk of overfitting.

Screenshots

In [14]:
```
# investigating the worst split of [evalC]
train_index, test_index = list(kf.split(X))[5]
X_train, X_test = X[train_index], X[test_index]
y_train, y_test = y[train_index], y[test_index]
count_vectorizer = CountVectorizer(lowercase=True, stop_words=stop_words)
count_vectorizer.fit(X_train)
vectorized_messages = count_vectorizer.transform(X_train).toarray()
gnb = GaussianNB()
gnb = gnb.fit(vectorized_messages, y_train)
result = gnb.predict(count_vectorizer.transform(X_test).toarray())
acc_score = accuracy_score(result, y_test.tolist())
print(acc_score)
from sklearn.metrics import confusion_matrix
print(df.label.unique())
print(confusion_matrix(y_test, result, labels=df.label.unique()))
```

```
0.7142857142857143
['income' 'living' 'private' 'standardOfLiving' 'leisure' 'finance']
[[1 0 0 0 0 0]
 [0 3 0 1 0 0]
 [0 0 1 0 0 0]
 [0 0 0 3 0 0]
 [0 1 0 4 7 0]
 [0 0 0 0 0 0]]
```

In [11]:
```
# [evalA]: preprocessed, omitting stop words
# evaluation using k-fold with k=10
X = df['preprocessed_data']
y = df['label']
kf = KFold(n_splits=10, shuffle=False)

# 16 stop words to omit
stop_words = ['aeu', 'all', 'byladembsbt', 'blz', 'bp', 'ccbadexxx', 'ga', 'gaa', \
              'geb', 'rfalld', 'spdudexxx', 'sskndexxx', 'to', 'ts', 'ue', 'vag']

accuracy_scores = []

for ind, data in enumerate(kf.split(X)):
    train_index, test_index = data
    print(f"{str(ind + 1).rjust(2)}/{kf.n_splits}:\t", end='')
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    count_vectorizer = CountVectorizer(lowercase=True, stop_words=stop_words)
    count_vectorizer.fit(X_train)
    vectorized_messages = count_vectorizer.transform(X_train).toarray()
    gnb = GaussianNB()
    gnb = gnb.fit(vectorized_messages, y_train)
    result = gnb.predict(count_vectorizer.transform(X_test).toarray())
    acc_score = accuracy_score(result, y_test.tolist())
    print(acc_score)
    accuracy_scores.append(acc_score)

print(f"mean:\t{mean(accuracy_scores)}")
```

```
 1/10:  1.0
 2/10:  1.0
 3/10:  0.8095238095238095
 4/10:  0.8095238095238095
 5/10:  0.7619047619047619
 6/10:  0.8095238095238095
 7/10:  0.9047619047619048
 8/10:  0.9047619047619048
 9/10:  1.0
10/10:  1.0
mean:   0.9
```

In [12]:
```
# [evalB]: no-preprocessing, no stop words
# evaluation using k-fold with k=10
X = df['accumulated_data']
y = df['label']
kf = KFold(n_splits=10, shuffle=False)

accuracy_scores = []

for ind, data in enumerate(kf.split(X)):
    train_index, test_index = data
    print(f"{str(ind + 1).rjust(2)}/{kf.n_splits}:\t", end='')
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    count_vectorizer = CountVectorizer(lowercase=True)
    count_vectorizer.fit(X_train)
    vectorized_messages = count_vectorizer.transform(X_train).toarray()
    gnb = GaussianNB()
    gnb = gnb.fit(vectorized_messages, y_train)
    result = gnb.predict(count_vectorizer.transform(X_test).toarray())
    acc_score = accuracy_score(result, y_test.tolist())
    print(acc_score)
    accuracy_scores.append(acc_score)

print(f"mean:\t{mean(accuracy_scores)}")
```

```
 1/10:  1.0
 2/10:  1.0
 3/10:  0.8095238095238095
 4/10:  0.8571428571428571
 5/10:  0.8095238095238095
 6/10:  0.9047619047619048
 7/10:  0.8571428571428571
 8/10:  1.0
 9/10:  1.0
10/10:  1.0
mean:   0.9238095238095239
```

In [13]:
```
# [evalC]: preprocessing, omitting stop words (including 'eu', 'eur' and 'euro')
# evaluation using k-fold with k=10
X = df['preprocessed_data']
y = df['label']
kf = KFold(n_splits=10, shuffle=False)

# 19 stop words to omit
stop_words = ['aeu', 'all', 'byladem sbt', 'blz', 'bp', 'ccbadexxx', 'eu', 'eur', 'euro', \
              'ga', 'gaa', 'geb', 'rfalld', 'spdudexxx', 'sskndexxx', 'to', 'ts', 'ue', 'vag']

accuracy_scores = []

for ind, data in enumerate(kf.split(X)):
    train_index, test_index = data
    print(f"{str(ind + 1).rjust(2)}/{kf.n_splits}:\t", end='')
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    count_vectorizer = CountVectorizer(lowercase=True, stop_words=stop_words)
    count_vectorizer.fit(X_train)
    vectorized_messages = count_vectorizer.transform(X_train).toarray()
    gnb = GaussianNB()
    gnb = gnb.fit(vectorized_messages, y_train)
    result = gnb.predict(count_vectorizer.transform(X_test).toarray())
    acc_score = accuracy_score(result, y_test.tolist())
    print(acc_score)
    accuracy_scores.append(acc_score)

print(f"mean:\t{mean(accuracy_scores)}")
```

```
 1/10:  1.0
 2/10:  1.0
 3/10:  0.7619047619047619
 4/10:  0.8095238095238095
 5/10:  0.7619047619047619
 6/10:  0.7142857142857143
 7/10:  0.9047619047619048
 8/10:  0.8095238095238095
 9/10:  1.0
10/10:  1.0
mean:   0.8761904761904762
```