Document

Background

TF-IDF

Assume that there are n terms in total. For term i in document j, we have $\text{TF-IDF}(i,j) = \text{TF}(i,j) \cdot \text{IDF}(i)$[1].

Term Frequency: $\text{TF}(i,j) = \frac{N_{ij}}{\sum_t N_{tj}}$, where N_{tj} is the number of term t appears in the document j.

Inverse Document Frequency: $\text{IDF}(i) = \log \frac{\text{Total number of documents}}{\text{Number of documents with term } i \text{ in it}}$

Then, we transform each document into a vector of length n, where the i-th bit of the j-th document is $\text{TF-IDF}(i,j)$.

Multinomial Naive Bayes

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set[2].

Bayes Theorem: $P(A|B) \cdot P(B) = P(B|A) \cdot P(A)$

Let N be the size of the vocabulary. Let the set of documents be denoted by D. Let the set of terms be denoted by T. Let the set of classes be denoted by C. Let F_{ic} be the sum of the TF-IDF value of term i of documents which belong to class c.

$$\hat{P}(t_n|c) = \frac{\alpha + F_{nc}}{N\alpha + \sum_{i=1}^{N} F_{ic}} \qquad \hat{P}(c) = \frac{\text{Total number of documents of class } c}{\text{Total number of documents}}$$

With given document t, we have:

$$P(c|t) = \frac{P(t|c) \cdot P(c)}{P(t)} \propto P(t|c) \cdot P(c) = \prod_{n=1}^{N} P(t_n|c) \cdot P(c) \propto \sum_{n=1}^{N} \log \hat{P}(t_n|c) + \log(\hat{P}(c))$$

Therefore, we can predict the class of document t by applying:

$$c = \text{argmax}_{c \in C}\, P(c|t)$$

Implementation

__init__

```python
def __init__(self, alpha=1e-2):
    self.feature_num = 0      # number of features/ number of terms
    self.label_num = 0     # number of labels
    self.log_prob = None     # log(P(t|c))
    self.label_log_prob = None     # log(P(c))
    self.alpha = alpha     # smoothing parameter, which can be set manually
```

train

```
def train(self, train_X, train_y):
    # init varaibles
    self.feature_num = train_X.shape[1]
    self.label_num = np.unique(train_y).size
    log_prob = np.zeros([self.label_num, self.feature_num])
    label_log_prob = np.zeros(self.label_num)
    # calculate probability tables
    for i in range(train_X.shape[0]):
        prob[train_y[i] - 1] += train_X[i]
        label_prob[train_y[i] - 1] += 1
    label_prob /= label_prob.sum()
    for i in range(self.label_num):
        prob[i] = (prob[i] + self.alpha) / (prob[i] + self.alpha).sum()
    # calculate log probability tables
    self.label_log_prob = np.log(label_prob)
    self.log_prob = np.log(prob)
```

test

```
def test(self, test_X, test_y):
    pred_y = np.zeros(test_y.shape[0])
    # c = argmax P(c|t) = argmax P(t|c)*P(c)
    for i in range(test_X.shape[0]):
        prob = np.dot(test_X[i], self.log_prob.T)
        pred_y[i] = np.argmax(self.label_log_prob + prob) + 1
    accuracy = (pred_y == test_y).sum() / test_y.shape[0]
    print('The accuracy in test set: {:.2f}%.'.format(accuracy*100))
```

Result

Applying $\alpha = 0.12$, the accuracy in test set is 63.98%.

Reference

[1] http://www.tfidf.com/

[2] https://en.wikipedia.org/wiki/Naive_Bayes_classifier