PA3

Env Setup

Use python 3.10.6 and poetry as dependency management

1. You may install poetry from this link
2. Run the following command under submitted directory.

```
poetry env use python
poetry install
poetry shell
```

3. You may run

```
python pa3.py
```

and see result.

- **Note: the `requirements.txt` file is auto generated by poetry through `poetry export -f requirements.txt -o requirements.txt --without-hashes`, you may try `pip install -r requirements.txt` but the environment can't be promised to be the same as mine.**

Source code logic

1. Load documents

Uses `os.listdir` to access all document files and sort them by filename through builtin `sorted` function.

2. Preprocess data

Uses `Preprocessor` written in `pa1` to preprocess the document, stopwords are `nltk.corpus.stopwords.words('english')`, stemmer is `nltk.stem.proter.ProterStemmer` and uses `[ \r\n,./\'"~!@#$%^&*()_`1234567890:;{}?\[\]+-]` as token delimiter.

3. Init NBClassifier and train

NOTE: logic of source code below are derived from pseudo code and formula in powerpoint provided by professor.

- Use chi-square feature selection, and source code is as below:

```
def feature_selection(self):
    if self.selection_type == 'chi':
        vocabulary = self.train_tfidf_vectorizer.get_terms()
        tf = self.tf_dict
        chi = {}
        for term in vocabulary:
            term_chi = 0
            for category in self.categories:
                n = np.zeros((2, 2))
                for i in range(0, 2):
                        for j in range(0, 2):
                            n[i][j] = sum([(term.term in tf[doc_id]) == j and (doc_id in
 self.category_to_doc_ids[category]) == i for doc_id in self.training_ids])
                N = n.sum()
                for i in range(2):
                    for j in range(2):
                        E = n.sum(axis=0)[j] * n.sum(axis=1)[i] / N
                        term_chi += (n[i][j] - E) ** 2 / N
            chi[term] = term_chi
        vocabulary = sorted(chi, key=chi.get, reverse=True)[:500]
        return vocabulary
    else:
        raise Exception('Invalid feature selection type')
```

After feature selection , vocabulary size reduced from `5212` to `500`.

- Train the model

```
def train(self):
    vocabulary = self.feature
    docs_count = len(training_ids)

    for category in self.categories:
        nc = len(self.category_to_doc_ids[category])
        self.prior[category] = nc / docs_count
        tct = {}
        tf = self.tf_dict
        for term in vocabulary:
            tokens_sum = 0
            doc_ids = self.category_to_doc_ids[category]
            tct[term.term] = sum([tf[doc_id][term.term] for doc_id in doc_ids if term.ter
m in tf[doc_id]])

        for term in vocabulary:
            self.cond_prob[term.term][category] = (tct[term.term] + 1) / (sum(tct.values
()) + len(vocabulary))
    return self
```

4. Predict the test dataset

```
def pred(self):
    doc_category = {}
    vocabulary = self.feature
    for doc_id in self.testing_ids:
        score = {}
        tf =self.tf_dict
        for category in self.categories:
            score[category] = self.prior[category] + sum([log(self.cond_prob[term.term][c
ategory]) * tf[doc_id][term.term] for term in vocabulary if term.term in tf[doc_id]])
        doc_category[doc_id] = max(score, key=score.get)
    return doc_category
```

5. Save prediction

Save the prediction information into `pred.csv` and the final f1-score is 0.97222.