We are required to implement the basic Naïve Bayes algorithm and make two improvements on it. The followings are what I have done:

**Preprocess.** First, I split the long sentences in each abstract into a list of words. Second, I deleted the stop words such as words 'the', 'a' and so on, which is actually common noise in the data.

**Build basic model.** As there are a large number of vocabularies in the data, we prefer to use a multinomial model, which uses word frequencies to represent the data instead of using the Boolean value to represent whether a word exits in the documents or not. Our prime aim is to calculate the posterior probability of giving a test text what the probabilities of being classified into different classes are, which is based on the Bayesian formula. Now, our aim can be separated into estimating the probability of likelihood and estimating the prior probability. Also, according to the independence assumption, the likelihood of giving a class what the probability of getting such text is can be calculated as the product of the likelihood of giving a class what the probability of getting each component word is. The following are the implements: **Step1**, we need to count (1) each word frequency in all training set, (2) the number of unique words in all training set, (3) each word frequency grouped by different classes, (4) the number of words in each class, and (5) the number of class labels in all instances. It's a good choice to store counts in dictionaries. **Step2**, we need to calculate the posterior probability. In order to get rid of the error of multiplying lots of tiny numbers, using logarithm to transfer the product to the sum as follows is a good way:

$$\mathrm{P}(\mathrm{Y}=y_k|X=X_i) \propto \log(\mathrm{P}(\mathrm{Y}=y_k)) + \sum_{j=1}^{n} \log(\frac{N(\mathrm{x}^j=\mathrm{x}^j,\mathrm{Y}=y_k)+1}{N(\mathrm{Y}=y_k)+1*J})$$

Where $y_k$ is a class label, $X_i$ is the given document, $\mathrm{x}^j, (j=1 \dots n)$ is any specific word in the document $X_i$, and $J$ is the number of unique words in the training set, which were counted in step1. **Step3**, we choose the class that has the largest posterior probability as prediction value.

**Build complement Naïve Bayes Model.** As the data is skewed with a large portion of class B and E ('A': 102, 'B': 1275, 'E': 1724, 'V': 99), in order to reduce the effect of having larger probability of being classified as the class with large proportion in all training instances, I used Complement Naïve Bayes model (CNB). The core of the CNB model is to use the data from not being in the class to estimate the probability. The only thing to modify in the code is the counts values used in posterior probability formula. Instead of adding the log-likelihood of being in the class, we subtract the log-likelihood of not being in the class.

$$\mathrm{P}(\mathrm{Y}=y_k|X=X_i) \propto \log(\mathrm{P}(\mathrm{Y}=y_k)) - \sum_{j=1}^{n} \log(\frac{N(\mathrm{x}^j=\mathrm{x}^j,\mathrm{Y}=\neg y_k)+1}{N(\mathrm{Y}=\neg y_k)+1*J})$$

**Transfer data by document frequency.** In order to keep the common words in the instances from dominating the classification, we can transfer the count of each word from 1 to a weight of document frequency $1*\log\left(\frac{\sum_k 1}{\sum_k \delta_{ik}}\right)$ as follows, where $\delta_{ik}=1$ if word $i$ exits in the training instance $k$.

$$\mathrm{P}(\mathrm{Y}=y_k|X=X_i) \propto \log(\mathrm{P}(\mathrm{Y}=y_k)) - \sum_{j=1}^{n} \log(\frac{N(\mathrm{x}^j=\mathrm{x}^j,\mathrm{Y}=\neg y_k)*\log\left(\frac{\sum_k 1}{\sum_k \delta_{ik}}\right)+1}{N(\mathrm{Y}=\neg y_k)*\log\left(\frac{\sum_k 1}{\sum_k \delta_{ik}}\right)+1*J})$$

In this way, if a word exists in the every document, it will have lower weight relatively.

**Results.** Finally we can estimate the prediction accuracy and variance based on ten cross-validations.

| | **Simple NB** | **CNB** | **Transfer CNB** |
|---|---|---|---|
| **Accuracy** | 0.94416 | 0.95933 | 0.96683 |
| **Variance (st error)** | 2.278e-05 | 2.38e-05 | 2.86e-05 |