Attached file: hw4.ipynb

The output file of the Persica database has been downloaded as a csv. Each row of information in the main table has 7 columns, and out of these 7 columns, I need two columns, title and text, as independent variables and category2 as a dependent variable. In the persica.csv file, the column information is written in one row. I read the lines one by one from the input of the file and keep lines 1, 2 and 6 (starting numbering from 0) as title, text and category, respectively. Finally, I merge the text and the title together and the result is an input_text column that will be used in the rest of the work. Also, in the category column, we have the target variable, which gives you the following 11 groups. The goal of the problem is to classify the texts into the following 11 classes.

{ 'علمی', 'بهداشتی', 'اقتصادی', 'تاریخی', 'اجتماعی', 'آموزشی', 'مذهب' , 'فقه و حقوق', 'فرهنگی', 'سیاسی'
'ورزشی', 'ی' }

Using sklearn.model_selection.train_test_split, I divide the data into two test and training grou ps with a ratio of 2 to 8. I will use 3 categories on this data and report the results of each one (p recision, recall, f-measure).
Before using the category, some processing should be done on the input text to the algorithm. The first part is related to cleaning the text and standardizing the text, which is stated in the des cription of Persica data that these pre-processings have been done, so we don't need it here. Th e second part is related to converting the text into a vector that can be used in classification alg orithms. The work steps are as follows:

1. Generating a word-document matrix by the word-sack method: I have a column for each word in the data set and I display a document as a row, and if any of these words are pr esent in it, the corresponding index is 1 and otherwise it will be 0. I do this step with the help of CountVectorizer model from sklearn.feature_extraction.text. So far, I have imple mented the simplest mode of converting text to vector.
2. Generating the tf-idf matrix from the previous step: instead of zero and one, we use the frequency of occurrence of words, so that the words that are rarely repeated in total, bu t appear more often in a certain text, become more important in recognizing the class of that document, and vice versa. . To generate this matrix, I use the TfidfTransformer mod el from sklearn.feature_extraction.text. The matrix that results from this step and the pr evious step have dimensions of 8799x60631, which means that the rows are equal to th e number of training data and the columns are equal to the number of words.
3. To reduce the dimension of the matrix space in which we work, I use the SVD method o n the vectors obtained from the previous step. This step yields vectors that have smaller dimensions than the previous 60631 step, but preserves maximum information by comp ression. This step is the implementation of the LSA section, which is implemented with t he TruncatedSVD model from sklearn.decomposition. Then I set the output vectors to 1 00 and it is clear that by increasing this number, the accuracy can be increased until the i ncrease is no longer observed.

4. I normalize the vectors such as the texts obtained in the previous step so that they all ha ve equal sizes and the effect of the vector size in determining the similarity between the texts is lost. This part is implemented with soft model 2 of Normalizer from sklearn.prep rocessing. In the first category (simple Bayes), due to the Bayesian nature of this model, the input data of the model could not be negative, so before entering step 5, I did anoth er scaling on the data so that all vectors with unit size have ranges between 0 and 1.
5. I define a classifier and train it for the x_train training matrix obtained from the previous steps. In this work, the three categories used are simple bayes, perceptron and SVM res pectively, all of which are built with sklearn classes. I used rbf kernel for SVM. After train ing each model, I ran a gridsearch, which is executed per parameter of the output dimen sion of LSA decomposition (SVD) for the numbers 100, 200, and 300. The goal was to see how much accuracy could be improved by increasing this dimension. For SVM, I also per formed this search for different kernels, including linear, polynomial and sigmoid. In all c ases, the number 300 gave better results, but its difference with dimension 100 in all mo dels was equal to 2 to 3% on all criteria, and the value of adding a dimension of 3 times i s not reasonable for this amount of accuracy. Then it is more time-consuming and bulky, which I think is more economical. For this reason, I perform the evaluation stage with th e same models that were made.

	Naive_Bayes	**Perceptron**	**SVM**
Precision by n_components = 100	0.77	0.76	*0.82*
Precision n_components = 300	0.80	0.78	*0.84*

6. I perform all the operations from 1 to 5 implemented as a pipeline for x_text texts, and for this obtained set, I perform the classification work with the model created in 5. The predicted labels are stored on y_predicted, and then with the precision_recall_fscore_support function of sklearn.metrics, I calculate the values of precision, recall and f-score in macro mode for the entire 11-class classification, the results of which are as follows.

	Naive_Bayes	**Perceptron**	**SVM**
precision	0.77	0.76	*0.82*
recall	0.78	0.78	*0.82*
f-score	0.77	0.76	*0.82*

The best model built for this 11-class classification problem here is clf_svm, a SVM classifier that uses the rbf kernel and works with vectors of size 100 as vectors such as LSA text. This model has a better condition than the other two models in terms of all 3 reported criteria.