1. Implementation

I've chosen to solve this homework in Python. I'm also using the libraries NumPy and SciPy. I've tried to follow the same pattern I've used in Homework 2. I have separate classes for each type of classifier. And I have classes for where I test each classifier. CrossValidatorTester where I test SVM, LogisticRegression and NaiveBayes. For Bagged forrest I have a different class (BaggedForestValidatorTester) due to the fact that it's tested totally different. And finally I have EnsembleValidatorTester class for testing the ensemble methods (SVM and LR over trees).

As data structure I initially started by storing the vectors inside Python dictionary in order to not consume too much memory (because each entry had 67k+ features). I created classes to calculate dot products, sums, multiplication between scalar and dictionary. Unfortunatelly using this approach, only training one epoch for one hyper parameter took 5 minutes on my laptop.

I then switch back to NumPy arrays (like I used in Homework 2). The computation time went down to 35-45 seconds for one epock. It was a big improvement, but the vectors were not sparsed.

Searching the internet, I found out that SciPy had matrixes classes that could store sparsed vectors. I switch to those and the computation time went down to 25 seconds per epoch. I decided to use this format for all the classifiers.

SVM and Logistic regression are the only classifiers where I used all the features from the data set. The results were similar between the two classifiers. For the others, the computation times were unrealistic using the current approach. SVM was slow so for detecting the best hyper parameter I was running only 1 epoch. For training the final classifier I used 5 epochs. For Logistic Regression, I've used 30 epoch for both steps.

For Naive Bayes, the things were not good. I was able to create the matrix with all the probabilities in a decent amount of time. The problem arised when I needed to predict the labels for the testing set. One prediction took like 2-3 seconds which is way too much. In order to get some data to report, I only used the first 200 features for Naive Bayes. And the results were not so good.

For Naive Bayes and Logistic regression I also loaded the values of -1 and 0. I was forced to do this in order to use some methods from the SciPy sparse matrix class that can count only non zero values efficiently. Using -1 and +1, I wouldn't have been able to use this advantageus method.

For Bagged Forest I took the Decision Tree class that I created for Homework 1 and changed it in order to use the SciPy matrixes. I wasn't able to create decision trees using all the 67k+ features, because the tree generation was too long. I decided to used only the first 200 features in order to get responses.

For the ensemble methods, I've used trees that were trained only for 200 features. Generating the trees and then transforming the results to feature vectors was also time consuming and becaused of that, I've saved the results in binary files so these steps can be skipped next time the main script is run. If the binary files are deleted, then all the steps are run.

2. Experiments results

	Best Hyper-parameters	Average cross-validation accuracy	Training accuracy	Test accuracy
SVM	Initial learning rate: 0.001 Regularization loss tradeoff: 10	80.27%	86.52%	80.43%
Logistic Regression	Initial learning rate: 0.01 Regularization loss tradeoff: 1	76.08%	79.56%	76.70%
Naive Bayes	Smoothing term: 2	67.18%	67.57%	65.21%
Bagged Forest	-	-	72.75%	71.81%
SVM over trees	Initial learning rate: 0.0001 Regularization loss tradeoff: 10	77.22%	76.69%	72.77%
Logistic Regression over trees	Initial learning rate: 0.01 Regularization loss tradeoff: 10	73.49%	71.11%	70.53%

Running the `main.py` for getting results like this might take between 30 and 50 minutes.