# COL 774: Assignment 2

**Due Date: 11:50 pm, March 11 (Sunday), 2018. Total Points: 59**

**Notes:**

- This assignment has two parts - Text Classification using Naïve Bayes and Handwritten digit classification using SVM.
- You should submit all your code (including any pre-processing scripts written by you) and any graphs that you might plot.
- Do not submit the datasets. Do not submit any code that we have provided to you for processing.
- Include a **single write-up (pdf) file** which includes a brief description for each question explaining what you did. Include any observations and/or plots required by the question in this single write-up file.
- You should use Python/MATLAB for all your programming solutions.
- Your code should have appropriate documentation for readability.
- You will be graded based on what you have submitted as well as your ability to explain your code.
- Refer to the course website for assignment submission instructions.
- This assignment is supposed to be done individually. You should carry out all the implementation by yourself.
- We plan to run Moss on the submissions. We will also include submissions from previous years since some of the questions may be repeated. Any cheating will result in a zero on the assignment, a penalty of -10 points and possibly much stricter penalties (including a **fail grade** and/or a **DISCO**).

1. **(26 points) Text Classification**
   In this problem, we will use the Naïve Bayes algorithm for text classification. The dataset for this problem is a subset of the IMDB movie review dataset and has been obtained from this website (look at the IMDB movie review dataset). Given a movie review, task is to predict the rating given by the reviewer. Read the website for more details about the dataset. You have been provided with separate training and test files containing 25,000 reviews (samples) each. Data is available at this link. A review comes from one of the eight categories (class label). Here, class label represents rating given by the user along with the review. You are provided four files i) Train text ii) Train labels iii) Test text iv) Test labels. Text files contain one review in each line and label files contain the corresponding rating.

   (a) **(10 points)** Implement the Naïve Bayes algorithm to classify each of the articles into one of the given categories. Report the accuracy over the training as well as the test set. In the remaining parts below, we will only worry about test accuracy.

   Notes:

   - Make sure to use the Laplace smoothing for Naïve Bayes (as discussed in class) to avoid any zero probabilities. Use $c = 1$.
   - You should implement your algorithm using logarithms to avoid underflow issues.
   - You should implement Naïve Bayes from the first principles and not use any existing Matlab/Python modules.

(b) **(2 points)** What is the test set accuracy that you would obtain by randomly guessing one of the categories as the target class for each of the articles (random prediction). What accuracy would you obtain if you simply predicted the class which occurs most of the times in the training data (majority prediction)? How much improvement does your algorithm give over the random/majority baseline?

(c) **(4 points)** Read about the confusion matrix. Draw the confusion matrix for your results in the part (a) above (for the test data only). Which category has the highest value of the diagonal entry? What does that mean? What other observations can you draw from the confusion matrix? Include the confusion matrix in your submission and explain your observations.

(d) **(4 points)** The dataset provided to is in the raw format i.e., it has all the words appearing in the original set of articles. This includes words such as 'of', 'the', 'and' etc. (called stopwords). Presumably, these words may not be relevant for classification. In fact, their presence can sometimes hurt the performance of the classifier by introducing noise in the data. Similarly, the raw data treats different forms of the same word separately, e.g., 'eating' and 'eat' would be treated as separate words. Merging such variations into a single word is called stemming.

- Read about stopword removal and stemming (for text classification) online.
- Use the script provided at this link to you to perform stemming and remove the stopwords in the training as well as the test data.
- Learn a new model on the transformed data. Again, report the accuracy.
- How does your accuracy change over test set? Comment on your observations.

(e) **(6 points)** Feature engineering is an essential component of Machine Learning. It refers to the process of manipulating existing features/constructing new features in order to help improve the overall accuracy on the prediction task. For example, instead of using each word as a feature, you may treat bi-grams (two consecutive words) as a feature. Come up with at least two alternative features and learn a new model based on those features. Add them on top of your model obtained in part (d) above. Compare with the test set accuracy that you obtained in parts (a) and parts (d). Which features help you improve the overall accuracy? Comment on your observations.

2. **(33 points) MNIST Handwritten digit Classification**
In this problem, we will use Support Vector Machines (SVMs) to build a handwritten digit classifier. We will be solving the SVM optimization problem using the Pegasos algorithm and also use a customized solver known as LIBSVM. You are provided with separate training and test example files. Each row in the (train/test) data file corresponds to an image of size 28x28, represented as a vector of grayscale pixel intensities followed by the label associated with the image. Every column represents a feature where the feature value denotes the grayscale value (0-255) of the corresponding pixel in the image. There is a feature for every pixel in the image. Last column gives the corresponding label. You are provided with a subset of the original MNIST dataset available at this link. We will work with this subset for the purpose of this assignment. For details of the original dataset, you are encouraged to look at this webpage.

(a) **(10 points)** Given a training dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$, recall that the (unconstrained) SVM optimization problem can be written as:

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^{m} \max(0, 1 - t_i) \tag{1}$$

where, $t_i = y^{(i)}(w^T x^{(i)} + b)$. Other symbols are as described in the class. Use mini-batch version of Pegasos algorithm described in
"Pegasos: Primal Estimated sub-GrAdient SOlver for SVM" to optimize above function and solve for $w, b$. You should start with Algorithm 1 given in the paper. Further, Algorithm 1 ignores the intercept term $b$. To incorporate $b$, use the description provided in Section 6 of the paper (use Equation 23). Your final implementation should compute the values of both $w$ and $b$. Use a batch size of 100 in your SGD implementation. Note that the paper uses a slightly different notation from the one covered in class and you should map the symbols accordingly. You should stick to the notation used in the class for consistency.

(b) **(5 points)** In class, we described the SVM formulation for a binary classification problem. In order to extend this to the multi-class setting, we train a model on each pair classes to get $\binom{k}{2}$ classifiers $k$ being the number of classes. During prediction time, we output the classifier which has the maximum

number of wins among all the $\binom{k}{2}$ classifiers. You can read more about one-vs-one classifier setting at the following link. Using your solver from part (a), implement one-vs-one multi-class SVM. Classify given MNIST dataset and report train and test accuracy for $C = 1.0$. In case of ties, choose the label with the highest digit value.

(c) **(8 points)** Now train a multi-class SVM on this dataset using the LIBSVM library, available for download from this link. Repeat part (b) using a linear Kernel as well as a Gaussian kernel with $\gamma = 0.05$ (i.e. $\gamma$ in $K(x, z) = \exp^{-\gamma * \|x-z\|^2}$). Use $C = 1.0$ in both cases i.e. 'linear' and 'Gaussian'. Report the test set accuracies for both linear as well as the Gaussian kernel setting. How do the accuracy of linear kernel compare with your implementation in part (b)? Comment.

(d) **(6 points)** Cross-validation is a technique to estimate the best value of the model parameters (e.g., $C$, $\gamma$ in our problem) by randomly dividing the data into multiple folds, training on all the folds but one, validating on the remaining fold, repeating this procedure so that every fold gets a chance to be the validation set and finally computing the average validation accuracy across all the folds. This process is repeated for a range of model parameter values and the parameters which give best average validation accuracy are reported as the best parameters. For a detailed introduction, you can watch this video. Use LIBSVM for this part, and you are free to use the cross-validation utility provided with the package.

In this problem, we will perform 10 fold cross-validation to estimate the value of the $C$ parameter for the Gaussian kernel case. We will fix $\gamma$ to be 0.05. Vary the value of $C$ in the set $\{10^{-5}, 10^{-3}, 1, 5, 10\}$ and compute the 10-fold cross-validation accuracy for each value of $C$. Also, compute the corresponding accuracy on the test set. Now, plot both the average validation set accuracy as well as the test set accuracy on a graph as you vary the value of $C$ on x-axis (you may use log scale on x-axis). What do you observe? Which value of $C$ gives the best validation accuracy? Does this value of the $C$ also give the best test set accuracy? Comment on your observations.

(e) **(4 points)** Draw a confusion matrix for your best results in part (d). Which class do you think is most difficult to classify? Visualize and report atleast three misclassified examples. Comment.

*Note: Since the features belong to the same metric, so you shouldn't normalize them to zero mean and unit variance as done in the first assignment. Rather, you should scale your features to [0,1] range for faster convergence.*