Random Forest Classifier Summary

To utilize a random forest classifier we first need a matrix with features to classify. To obtain such a matrix we use a bag of words approach on the text documents. What a bag of words does is assigns a fixed integer value to every word across all documents. We then obtain a dictionary of words. Now for a document with ID i we take the frequency of each word and store it in the ijth position of a matrix, where j represents the fixed integer value of the word in the dictionary. It is important to note that this will be a sparse matrix since each text document will contain some amount of unique words. At this point we have a matrix which can tell us about the feature of a text document consisting of total word counts for any given word, but to be more useful we would like to determine the density of a given word in a document as well as its uniqueness to that document.

We now transform our matrix using a term frequency – inverse document frequency (tf-idf) method to account for these desired features. To compute the tf-idf of a word in a document we take $tf(w)$ to be the amount of times w appears in a document and $idf(w,d) = \log\left(\frac{n}{df(w,d)}\right) + 1$ where n is the total number of documents and $df(w,d)$ represents the amount of documents that contain w, and finally we take tf-idf to be $tf(w) \times idf(w,d)$. In a sense when we create the tf-idf matrix we take our matrix obtained from a bag of words and multiply the nonzero values by their respective $idf(w,d)$ value. Once we obtain a matrix consisting of tf-idf values we will have a useful feature that to consider in a random forest classifier.

Before we discuss a random forest it is first useful to introduce the notion of a decision tree. A decision tree is essentially a process which separates data based on some quality. A very simple example would be if one were to have a set of points in $\mathbb{R}^2$ with some specified class and we would like to separate them as best we can. Let's take class 1: $\{(-1,-2),(1,2)\}$, class 2: $\{(3,-3),(2.5,1)\}$, class 3: $\{(1,-4),(4,5)\}$, and class 4: $\{(-5,1)\}$. If a decision tree can only make vertical and horizontal lines then it will most likely first create a line such as $x = -4$ which will separate class 4 from each other class. In this case we have that the left side of this separation is "pure." Now any slice of the plane will have to cut off two points in the same class. The classifier will do each of the possible divisions to determine which has the least loss. It will keep making divisions until each section is "pure", meaning it contains only one class, or we specify some depth. Once we classify the data any new data point will fall into one of these sections and if it were to fall somewhere to the left of $x = -4$, for example, the classifier would likely predict that that point is in class 4.

In our situation a decision tree will take a row from the tf-idf matrix and use the tf-idf score as a distinguishing feature. Obviously we have 9 classes and text documents consisting of thousands of words so the decision tree must make distinctions using a hyperplane instead of lines in $\mathbb{R}^2$. In a sense we can still use the example from before as a framework but in this case a decision tree will consider points in much higher dimensions.

Finally we consider a random forest classifier. What occurs in the classification process is we take our training data, which in this case is the tf-idf matrix where each row of corresponds to a text document, and repeatedly take a random subset of it (with replacement) and create a decision tree for each random subset. Moreover each decision tree, at each split, selects a random subset of the features to further split. What we end up with is a large amount of decision trees (a so-called “forest”) where each classifies the data in a slightly (or perhaps not so slightly) different way.

When it comes to classifying new data what will occur is that each tree in the random forest will vote on what class it thinks the data belongs too. The classifier then takes a majority rules approach and predicts the class in that way. In theory this approach should avoid overfitting the data, and that makes it one of the more prominent classifiers currently in use.