HW5 Starter

Problem 1: Support Vector Machines and Kernels

1a: Describe a setting of the hyperparameters `gamma` and `C` that is sure to *overfit* to any typical training set and achieve zero training error.

As gamma and C tend towards infinity, overfitting is bound to occur

1b: Describe a setting of the hyperparameters `gamma` and `C` that is sure to *underfit* to any typical training set.

As gamma and C tend towards 0, underfitting is bound to occur

1c: If you were performing grid search to find the SVM that generalized best to unseen data, what range of values for gamma, C would you recommend? Specify your answer with two lines of NumPy code (e.g. using a list like [1, 2, 3] or np.linspace or np.logspace). Also provide 1-2 sentences of justification.

C = [0.0001,0.01,1,100,10000]

gamma = [$(1/numfeatures)^{-4}$,$(1/numfeatures)^{-2}$,$(1/numfeatures)$,$(1/numfeatures)^{2}$,$(1/numfeatures)^{4}$]

I obtained the median values from sklearn's default value when calling SVC. Higher and lower tested values are based off of practical suggestions that hyperparameters should increase/decrease exponentially during tuning.

1d: Answer the following True/False questions, providing *one sentence* of explanation for each one

1d(i): When used for binary classification, Support Vector Machines (SVMs) provide an estimate of the probability that a given feature vector x_i will have a positive label: $p(Y_i = 1|x_i)$.

False. SVM uses calculated boundaries and margins to predict binary classifications and is not based on probability

1d(ii): An advantage of an SVM is that the optimal weight vector w (a vector with one entry per feature) will typically be sparse.

False. Although SVM will benefit from having a sparse weight vector, sparse optimal weight vectors are never guranteed.

1d(iii): When choosing a kernel function k, it should be the case that for any finite dataset of N distinct examples, the $N \times N$ kernel matrix is invertible.

True. The symmetric property of kernels is only guranteed if the matrtix is invertible

Problem 2: Random Forests

2a: Describe a setting of the hyperparameters `n_estimators`, `max_features`, and `min_samples_leaf` that is sure to *overfit* to any typical training set. You can assume no BAgging (`bootstrap=False`).

Such a setting would have a high value for n_estimator (10^6 or higher), low min samples_leaf (1), and high max_features (n, where n = no. features)

2b: Describe a setting of the hyperparameters `n_estimators`, `max_features`, and `min_samples_leaf` that is sure to *underfit* to any typical training set. You can assume no BAgging (`bootstrap=False`).

Such a setting would have a low value for n_estimator (1), high min samples_leaf (n, where n = no. features), and low max_features (1)

2c: If you were performing grid search to find the RandomForest that generalized best to unseen data, what range of values for n_estimators, max_features, and min_samples_leaf would you recommend? Should we set bootstrap to True or False? Specify your answer with a few lines of NumPy code (e.g. using a list like [1, 2, 3] or np.linspace or np.logspace to set each variable). Include a sentence or two of justification. Assume that you can't afford more than 5 distinct values for each variable.

n_est = $[1, 10, 100, 1000, 10000]$ max feat = $[1, n/4, n/2, 3n/4, n]$ min samp = $[1, n/4, n/2, 3n/4, n]$ bootstrap = True

I chose the values of n_est from 1 to 10000 as 1 is the minimum and higher than 10000 would cause face performance runtime issues, the features and min sample_leaf were decided based on splitting n into 5 quadrants of values that could be tested and fine-tuned once the boundary for overfitting and underfitting is decided. bootstrap is set to true as it would help reduce variance and would usually provide better predictions.

2d: Answer the following True/False questions, providing *one sentence* of explanation for each one:

2d(i): When used for binary classification, RandomForests (RFs) can provide an estimate of the probability that a given feature vector x_i will have a positive label: $p(Y_i = 1|x_i)$.

True. When used for binary classification, Random forests provide the average probability that features vector x(i) has a positive label determined by all the decision trees constructed.

2d(ii): With bootstrap aggregating enabled, random forests will almost always severely overfit the training data if the number of trees used (e.g. the `n_estimators`) is very large (say, over 500).

False. Whether value of n_estimators used overfits is determined by the number of features in the feacture vector as well as the maximum depths of trees in the Random Forest.

2d(iii): When fitting random forests, it is generally a good idea to allow each node of each tree to consider as many features as possible (e.g. `max_features` should be large).

False. Overfitting can occur if all features are used. This process is also expensive

2d(iv): Random forests only use randomness when creating many similar datasets via BAgging. No other step of the algorithm uses randomness.

False. Assuming that not all features are considered when splitting, trees also randomly sample features

Problem 3

3a: Can we apply ROC curves to binary classifiers that cannot easily produce probabilities $\hat{p}_i \in (0, 1)$, but produce some real-valued scores $s_i \in \mathbb{R}$ for each example? How would we select the range of thresholds to evaluate? ¶

Yes, we can still apply ROC curves as TPR and FPR used in plotting ROC curves do not need to be based off of probabilities. In order to select range of thresholds, we can look at the minimum and maximum value of real-valued scores that we obtain and select and even spread.

3b: Suppose you fit a classifier to data, and you observe a TPR of 0.3 and an FPR of 0.7. Your friend says that this is worse than a random classifier (if plotted on an ROC curve, would fall below the TPR=FPR diagonal line), and so you should throw this result away. Is there anything better you can do? Describe how to tranform the predicted binary labels to reach better performance. What TPR and FPR would you expect to achieve?

A TPR of 0.3 and FPR of 0.7 means that our classifier tends to produce opposite predictions from the actual value. We can simply apply the opposite of our classifier's predictions to obtain a better-than-random set of predictions with TPR of 0.7 and FPR of 0.3.