Sit720 9.1D Distinction Task – I

Question 1:

The Voting Classifier combines predictions from multiple base models (like Random Forest, AdaBoost etc) in two main ways:

1. Majority Class Lables (Hard Voting):
   - Mechanism: In hard, voting each base model predicts a class label. The final prediction is made by selecting the class label that is most frequently predicted by the base models.
   - Example: Suppose Classifier 1 predicts class 1, Classifier 2 predicts class 1 and Classifier 3 predicts class 2, using hard voting, the final prediction would be class 1 because it has the majority of votes. If there is a tie the Voting Classifier selects the class based on the ascending order (i.e. picks first one).
2. Weighted Average Probabilities (Soft Voting):
   - In soft voting each base model provides predicted probabilities for each class. The final prediction is derived from the class label with the highest average probability for each class.
   - Example: Suppose Classifier 1 has [0.2, 0.5, 0.3] weights for each class, Classifier 2 has [0.6, 0.3, 0.1] and Classifier 3 has [0.3,0.4,0.3]. Treating all classifiers with equal weight the averages for each class would be: Class 1 = (0.2+0.6+0.3)/3 = 0.37, Class 2 = (0.5+0.3+0.4)/3 = 0.4, Class 3 = (0.3+0.1+0.3)/3 = 0.23. Class 2 would be the final prediction as it has highest probability.

Strategies for Conflict Resolution:

1. Tie-Breaking Rule for hard voting as mentioned above.
2. For soft voting apply different weights to classifiers based on their performance. More accurate models have a greater influence on the final prediction.
3. Hyperparameter Tuning: Use techniques like GridSearchCV with VotingClassifier to tune hyperparameters of individual models to improve performance.
4. Combine diverse Classifiers: Using different diverse models such as decision tress, logistic regressions and SVM can reduce likelihood of conflicting predictions as it combines the strengths and weaknesses of each model, providing a more balanced final prediction.

Question 2:

SVMs are primarily for binary classification (yes or no type tasks) but can effectively perform multiclass classification through two main approaches.

1. One-vs-One (OvO)

This involves training a binary SVM classifier for every pair of classes. If you have k classes that implies you train k(k-1)/2 binary classifiers. During prediction, each classifier votes for a class and the class with most votes is selected.

This method can be effective if the number of classes is relatively small. It can become computationally expensive as the number of classes increases.

2. One-vs-Rest (OvR)

This involves training a binary SVM classifier for each class, with the class being considered as the positive class and all other classes as the negative class. For k classes you will train k classifiers. During prediction the classifier that provides the highest score is selected.

This method is simpler and scales better to a larger number of classes compared to OvO. It may be less effective if classes are not well separated, as each classifier is trained with a large number of negative samples.

To successfully implement each, it requires correct choice of kernel (such as RBF commonly), hyperparameter tuning (such as cost parameter and kernel parameters) and scaling features (normalisation or standardisation is often needed).

References for Q1 and 2

Scikit-learn. (n.d.). *Support vector machines*. Retrieved July 24, 2024, from https://scikit-learn.org/stable/modules/svm.html#support-vector-machines

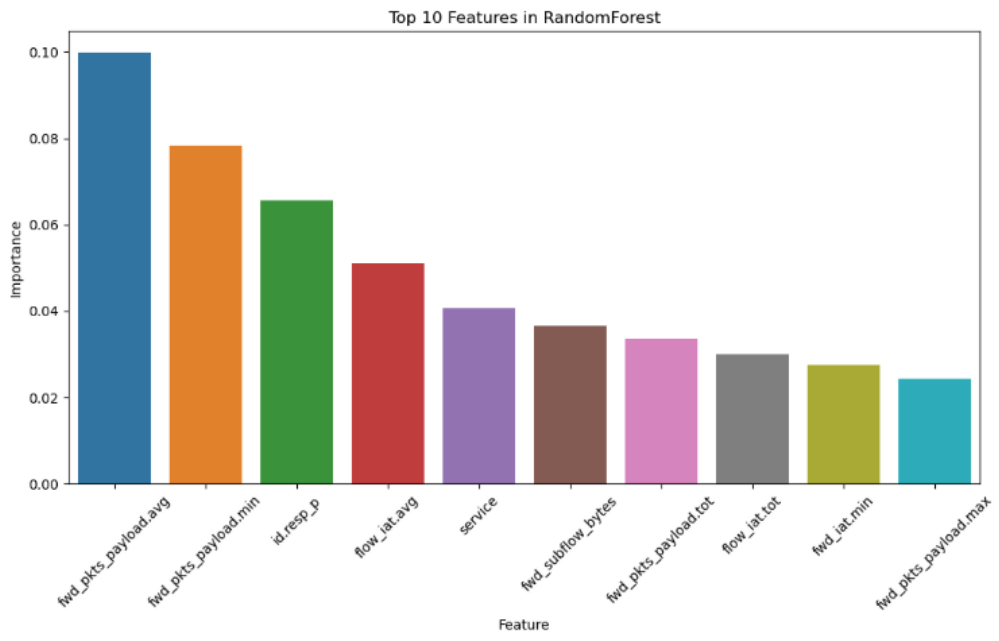
Scikit-learn. (n.d.). *Voting classifier*. Retrieved July 24, 2024, from https://scikit-learn.org/stable/modules/ensemble.html#voting-classifier

Question 3:

For this question we analyse the importance of features in predicting target using two different approaches.

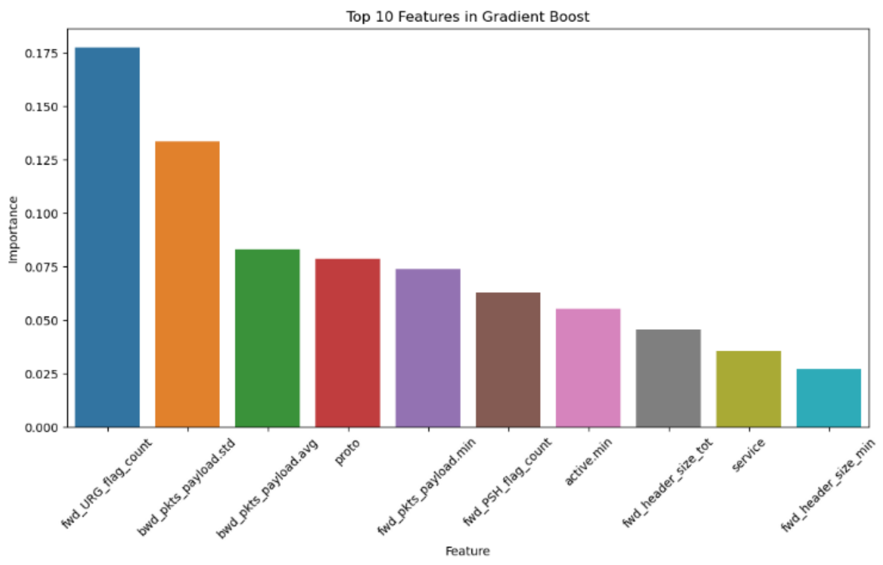
Method 1: Random Forest

Results of Random Forest feature importance are summarised in this bar plot.



Method 2: Gradient Boost

Results of Gradient Boost feature importance are summarised in this bar plot.



RandomForest importance shows fwd_pkts_payload.avg is the most importance feature in predicting Target with 0.10 importance, followed by fwd_pkts_payload.min with 0.08 importance.

Gradient Boost method has found different importances with higher values. The most important feature being fwd_URG_flag_count with 0.18 importance, followed by bwd_pkts_payload.std with 0.13 importance.

For each method we got different features being most important with different importance scores. This is due to how each of the feature importances is calculated.

In Random Forest feature importance is computed as the total reduction in criterion from that feature, known as the Gini importance.

For Gradient Boost the feature importance is averaged over all targets from all tress it makes.

The different approaches to how each method calculates feature importance is why we get different results.

References for Q3

XGBoost Developers. (n.d.). *XGBoost Python API documentation*. XGBoost. Retrieved July 24, 2024, from https://xgboost.readthedocs.io/en/latest/python/python_api.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). *RandomForestClassifier* (version 0.24.2) [Software]. Scikit-learn. Retrieved July 24, 2024, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Question 4:

For question we were tasked with building 3 supervised ML models except ensemble models. For model 1 I chose Stochastic Gradient Descent (SGD), for model 2 I choose Logistic Regression and for model 3 I chose Multi-Layered Perceptron (MLP) classifier.

(a) Model Performance Summary using Train and Test accuracy

For model 1the original default values gave a test accuracy of 0.856 and training accuracy of 0.853. Doing hyperparameter tuning with 5 different variations yielding in no improvement. I will discuss hyperparameter tuning in part (c).

For model 2 the original default values gave a test accuracy of 0.808 and training accuracy of 0.807. Doing hyperparameter tuning with 3 different variations yielding in a boost to accuracy, with test accuracy of 0.919 and train accuracy of 0.917.

For model 3 the original default values gave a test accuracy of 0.915 and training accuracy of 0.917. Doing hyperparameter tuning with 8 different variations yielding in a boost to accuracy, with test accuracy of 0.954 and train accuracy of 0.953.

The presented results were not overfitted. The train and test accuracies were basically the same or didn’t differ by much. I purposely got both accuracies to test for overfitting.

(b) Model Design Justification

I used SGD for its efficiency in handling large datasets and its ability to update model weights more frequently.

I chose Logistic Regression for its simplicity and effectiveness in binary classification with extension into multi-class classification available.

I picked MLP for its ability to capture non-linear relationships through hidden layers and activation functions .

(c) Hyperparameter tuning

For model 1 I changed the loss and alpha parameters. Choosing from loss = {hinge, square_hinge, modified_huber}, alpha = {0.0001, 0.001,0.01}. The hinge function gives a linear SVM, squared_hinge is hinge with a quadratic penalty, and modified huber is a smooth loss function that brings tolerance to outliers. Alpha is the regularisation factor. By tuning the loss function and regularisation factor I wasn't able to get a model that reflects the data better. The best model was loss = hinge and alpha = 0.0001.

For model 2 I changed the C and mutli_class parameters. I choose from C = {0.1,1}, multi_class = {auto, 'ovr'}. Mult class allows us to change from binary to multi class by using ovr. C is the inverse of regularisation strength. Only changing multiclass to ovr had biggest impact as this is a multiclass target. Changing C had no effect.

For model 3 I tuned hidden_layer_sizes, activation and solver. I choose from hidden_layer_sizes = {50,100,200}, activation = {identity, logistic, tanh, relu} and solver = {lbfgs, sgd, adam}. Hidden layer sizes in the number of neurons in the ith hidden layer, activation is the activation function for hidden layer and solver is the solver for weight optimization. The best result came from 100 hidden layer sizes, activation = tanh and solver = adam.

(d) Recommendations

Based on accuracy results for training and test data (ranging from 0.808 to 0.954) the best model was MLP using tanh activation with 100 hidden layers and adam solver. It appears this model was best in capturing the underlying patterns in this complex dataset. It also shows importance of hyperparameter tuning and model selection. Doing both resulted in a near 20% accuracy increase.

References for Q4

scikit-learn developers. (n.d.). *sklearn.neural_network.MLPClassifier*. Retrieved July 24, 2024, from https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

scikit-learn developers. (n.d.). *sklearn.linear_model.SGDClassifier*. Retrieved July 24, 2024, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

scikit-learn developers. (n.d.). *sklearn.linear_model.LogisticRegression*. Retrieved July 24, 2024, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Question 5:

a) When are ensemble models better?

Based from results in Q4 and Q5 I can infer ensemble models such as random forest are better when we have more complex datasets with lots of features. Model performance in Q5 was 0.998 test accuracy for all ensemble methods used. This was much higher than all supervised learning methods used in Q4. This shows they were better able to capture the patterns in the dataset.

b) Similarities/Differences for Ensemble methods used

How each method works:

Random Forest: Fits a number of decision tree classifiers over multiple subsets of the data and averages to improve predicting accuracy and control overfitting.

Bagging classifier: Fits base classifiers each on random subsamples of dataset and aggregates their individual predictions (either by voting or averaging).

From these methods we see they are quite similar in how they break dataset into multiple subsets and aggregates their predictive power. The difference is Random Forest aggregates over multiple subsets and bagging aggregates individually.

Voting classifier combines multiple classifiers and weights their predictions to pick best fit. It balances the strengths and weaknesses of each classifier.

c) Preferable method

All ensemble methods used produced exact same test accuracy (0.998) and train accuracy (1.0). Based on this they are all great options to use. I would suggest using all with voting classifier to aggregate all classifiers would be the best option to weigh the strengths and weaknesses of each.

d) Performances of Q4 vs Q5

All ensemble methods has 0.998 test accuracy and 1.0 train accuracy. The highest test accuracy in Q4 was MLP (activation = tanh, hidden layer sizes = 100, solver = adam) with 0.954 test accuracy and 0.953 train accuracy. The ensemble methods outperformed the supervised ML methods. MLP was close due to how the complexity of multiple hidden layer networks operate, however, the complexity of the data was better suited to multiple decision tree methods.

Both the default setting of Random Forest and BaggingClassifier were used. To pick the best overall model we need to see which has base parameters that are less complex:

sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None, monotonic_cst=None) class sklearn.ensemble.BaggingClassifier(estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0)

These are the default settings for each method. BaggingClassifier uses fewer base models (10 vs 100), it doesn't enforce complex criteria for splits or max depths and uses fewer hyperparameters. Hence, Bagging is the best model based on being least complex and equal best performance.

e) Building other ensemble model other than decision tree

Voting Classifier used here is a non- decision tree ensemble method. It can combine three non-decision tree methods such as SVM, KNN, and Logistic Regression to make its predictions.

References for Q5

Scikit-learn developers. (n.d.). sklearn.ensemble.RandomForestClassifier. Retrieved July 24, 2024, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Scikit-learn developers. (n.d.). sklearn.ensemble.BaggingClassifier. Retrieved July 24, 2024, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

Scikit-learn developers. (n.d.). sklearn.ensemble.VotingClassifier. Retrieved July 24, 2024, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html