Background:
Ensembles are methods that combine several machine learning models to make one powerful model that is better than all the models that it is comprised of. The ensemble can consist of models of different types (e.g.: decision trees, neural nets, k-nearest neighbor,..) or they can also be created by combining models of the same type with different hyper-parameter values. In either case, the individual models should have two basic characteristics: an error rate below 0.5 (weak learners) and should be diverse. The reason for the first one is that we want our ensemble consist of models that perform better than random guessing. And the reason for the second one is that we need models that make different mistakes – if the different models perform well on different data points then combining them together will yield a model that is overall better than the individual ones.

There are different types of ensemble methods: bagging, random forest, or boosting.
The bagging method first takes the training data set of size n and creates multiple bootstrapped subsamples of size n by doing sampling with replacement. Therefore, some subsamples may contain duplicate data points or may be missing some data points that are in the original training set. The second step in bagging is to create a model based on each subset and then aggregate the results of these models through voting (weighted or unweighted) to output the final predictor.
Random forest uses the same procedures as bagging except for the subsets based on which the base models are trained don't contain all the features – instead, features are randomly selected. This ensures that the base models are different from each other.
Boosting is also similar to bagging, but the base learners are trained in sequence on a weighted versions of the training dataset. In particular, the first base learned is trained with equal weights. Then the weights are updated based on the errors the first base learner made – more weight is assigned to misclassified examples. Then the second base learner is trained on that weighted data and so on.

In this lab, I implemented several ensemble classifiers using Python's Sklearn library and applied it to a Blood Transfusion dataset from the UC Irvine Machine Learning Repository. The dataset contains 748 donor data and has 4 features: R (Recency - months since last donation), F (Frequency - total number of donation), M (Monetary - total blood donated in c.c.), T (Time - months since first donation), and a binary variable representing whether he/she donated blood in March 2007 (1 stand for donating blood; 0 stands for not donating blood).

Task 1:
In task 1, I made a random forest ensemble model using Sklearn's ensemble.RandomForestClassifier. It consists of many decision trees that are grown to the largest extent possible. I also created an

Adaboost ensemble classifier with sklearn.ensemble.AdaBoostClassifier, which uses decision stumps (trees with 1 node) as base classifiers. I also experimented with different hyper-parameters and tuned them using GridSearchCV.
Both ensemble models (random forest and adaboost) performed well in terms of overall accuracy on training and testing data. The random forest model achieved 94% and 76% accuracy on training and testing data respectively. The adaboost model had 81% accuracy on both testing and training data. Also, both ensemble models performed very well at predicting that a person did not donate blood. The true negative rate on testing data was 89% and 95% for random forest and adaboost respectively. However, the models did not perform too well at predicting that a person will donate blood. The true positive rate was 25% and 30% for random forest and adaboost respectively. Perhaps the reason for that is that there were not as many positive examples as there were negative. Therefore, the model could be improved if we could add more training examples or did further hyper-parameter tuning. The experiments with different hyper-parameters and best models are reported below.

Random Forest Model:

```
      trainAvg     trainSD    testAvg    testSD    fitTime  parameters
--  ----------  ----------  ---------  --------  ---------
------------------------------------------------------------------------------------------------------
 0    0.930086  0.0199364    0.483221  0.148706  0.0481192  {'class_weight': 'balanced', 'criterion': 'gini', 'oob_score': True, 'n_estimators': 30}
 1    0.930086  0.0199364    0.483221  0.148706  0.0455276  {'class_weight': 'balanced', 'criterion': 'gini', 'oob_score': False, 'n_estimators': 30}
 2    0.937929  0.018022     0.478747  0.150883  0.0625323  {'class_weight': 'balanced', 'criterion': 'gini', 'oob_score': True, 'n_estimators': 35}
 3    0.937929  0.018022     0.478747  0.150883  0.0523984  {'class_weight': 'balanced', 'criterion': 'gini', 'oob_score': False, 'n_estimators': 35}
 4    0.932886  0.020081     0.474273  0.148102  0.063005   {'class_weight': 'balanced', 'criterion': 'gini', 'oob_score': True, 'n_estimators': 45}
 5    0.932886  0.020081     0.474273  0.148102  0.0531805  {'class_weight': 'balanced', 'criterion': 'gini', 'oob_score': False, 'n_estimators': 45}
 6    0.931765  0.0201341    0.47651   0.148134  0.0643397  {'class_weight': 'balanced', 'criterion': 'gini', 'oob_score': True, 'n_estimators': 50}
 7    0.931765  0.0201341    0.47651   0.148134  0.0577914  {'class_weight': 'balanced', 'criterion': 'gini', 'oob_score': False, 'n_estimators': 50}
 8    0.938488  0.0184178    0.47651   0.148134  0.0694961  {'class_weight': 'balanced', 'criterion': 'gini', 'oob_score': True, 'n_estimators': 55}
 9    0.938488  0.0184178    0.47651   0.148134  0.0618143  {'class_weight': 'balanced', 'criterion': 'gini', 'oob_score': False, 'n_estimators': 55}
10    0.940164  0.0178724    0.474273  0.14523   0.122667   {'class_weight': 'balanced', 'criterion': 'gini', 'oob_score': True, 'n_estimators': 100}
11    0.940164  0.0178724    0.474273  0.14523   0.109788   {'class_weight': 'balanced', 'criterion': 'gini', 'oob_score': False, 'n_estimators': 100}
12    0.930086  0.0199364    0.47651   0.147718  0.0454871  {'class_weight': 'balanced', 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 30}
13    0.930086  0.0199364    0.47651   0.147718  0.042038   {'class_weight': 'balanced', 'criterion': 'entropy', 'oob_score': False, 'n_estimators': 30}
14    0.937928  0.018112     0.467562  0.149818  0.0525614  {'class_weight': 'balanced', 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 35}
15    0.937928  0.018112     0.467562  0.149818  0.0481924  {'class_weight': 'balanced', 'criterion': 'entropy', 'oob_score': False, 'n_estimators': 35}
16    0.932886  0.020081     0.467562  0.146477  0.0667179  {'class_weight': 'balanced', 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 45}
17    0.932886  0.020081     0.467562  0.146477  0.0611568  {'class_weight': 'balanced', 'criterion': 'entropy', 'oob_score': False, 'n_estimators': 45}
18    0.932324  0.0205049    0.463087  0.147709  0.0747174  {'class_weight': 'balanced', 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 50}
19    0.932324  0.0205049    0.463087  0.147709  0.0721787  {'class_weight': 'balanced', 'criterion': 'entropy', 'oob_score': False, 'n_estimators': 50}
20    0.939047  0.0186218    0.463087  0.147709  0.0860049  {'class_weight': 'balanced', 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 55}
21    0.939047  0.0186218    0.463087  0.147709  0.0784983  {'class_weight': 'balanced', 'criterion': 'entropy', 'oob_score': False, 'n_estimators': 55}
22    0.940164  0.0178724    0.472036  0.1444    0.150356   {'class_weight': 'balanced', 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 100}
23    0.940164  0.0178724    0.472036  0.1444    0.13504    {'class_weight': 'balanced', 'criterion': 'entropy', 'oob_score': False, 'n_estimators': 100}
24    0.931205  0.0206756    0.478747  0.149305  0.0505303  {'class_weight': 'balanced_subsample', 'criterion': 'gini', 'oob_score': True, 'n_estimators': 30}
25    0.931205  0.0206756    0.478747  0.149305  0.0468919  {'class_weight': 'balanced_subsample', 'criterion': 'gini', 'oob_score': False, 'n_estimators': 30}
26    0.937371  0.0178636    0.47651   0.150876  0.0594429  {'class_weight': 'balanced_subsample', 'criterion': 'gini', 'oob_score': True, 'n_estimators': 35}
27    0.937371  0.0178636    0.47651   0.150876  0.0545195  {'class_weight': 'balanced_subsample', 'criterion': 'gini', 'oob_score': False, 'n_estimators': 35}
28    0.932886  0.020081     0.469799  0.149121  0.0757825  {'class_weight': 'balanced_subsample', 'criterion': 'gini', 'oob_score': True, 'n_estimators': 45}
29    0.932886  0.020081     0.469799  0.149121  0.0717679  {'class_weight': 'balanced_subsample', 'criterion': 'gini', 'oob_score': False, 'n_estimators': 45}
```

```
30    0.937926  0.0182024    0.469799  0.148106  0.0838116  {'class_weight': 'balanced_subsample', 'criterion': 'gini', 'oob_score': True, 'n_estimators': 50}
31    0.937926  0.0182024    0.469799  0.148106  0.0787371  {'class_weight': 'balanced_subsample', 'criterion': 'gini', 'oob_score': False, 'n_estimators': 50}
32    0.938488  0.0184178    0.469799  0.149854  0.0940451  {'class_weight': 'balanced_subsample', 'criterion': 'gini', 'oob_score': True, 'n_estimators': 55}
33    0.938488  0.0184178    0.469799  0.149854  0.0862508  {'class_weight': 'balanced_subsample', 'criterion': 'gini', 'oob_score': False, 'n_estimators': 55}
34    0.940164  0.0178724    0.474273  0.144294  0.167776   {'class_weight': 'balanced_subsample', 'criterion': 'gini', 'oob_score': True, 'n_estimators': 100}
35    0.940164  0.0178724    0.474273  0.144294  0.157838   {'class_weight': 'balanced_subsample', 'criterion': 'gini', 'oob_score': False, 'n_estimators': 100}
36    0.930645  0.0202802    0.485459  0.145223  0.0611247  {'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 30}
37    0.930645  0.0202802    0.485459  0.145223  0.0576282  {'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'oob_score': False, 'n_estimators':
30}
38    0.937928  0.018112     0.474273  0.145576  0.07924    {'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 35}
39    0.937928  0.018112     0.474273  0.145576  0.0714223  {'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'oob_score': False, 'n_estimators':
35}
40    0.932326  0.0196418    0.478747  0.145088  0.0892557  {'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 45}
41    0.932326  0.0196418    0.478747  0.145088  0.0820044  {'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'oob_score': False, 'n_estimators':
45}
42    0.937926  0.0182024    0.472036  0.145509  0.103915   {'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 50}
43    0.937926  0.0182024    0.472036  0.145509  0.0939026  {'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'oob_score': False, 'n_estimators':
50}
44    0.939047  0.0186218    0.472036  0.145509  0.109531   {'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 55}
45    0.939047  0.0186218    0.472036  0.145509  0.101794   {'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'oob_score': False, 'n_estimators':
55}
46    0.940164  0.0178724    0.478747  0.143802  0.196612   {'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'oob_score': True, 'n_estimators':
100}
47    0.940164  0.0178724    0.478747  0.143802  0.185882   {'class_weight': 'balanced_subsample', 'criterion': 'entropy', 'oob_score': False, 'n_estimators':
100}
48    0.948547  0.0082128    0.49217   0.137767  0.0416131  {'class_weight': None, 'criterion': 'gini', 'oob_score': True, 'n_estimators': 30}
49    0.948547  0.0082128    0.49217   0.137767  0.0372499  {'class_weight': None, 'criterion': 'gini', 'oob_score': False, 'n_estimators': 30}
50    0.951347  0.00874908   0.494407  0.13688   0.0476902  {'class_weight': None, 'criterion': 'gini', 'oob_score': True, 'n_estimators': 35}
51    0.951347  0.00874908   0.494407  0.13688   0.0434758  {'class_weight': None, 'criterion': 'gini', 'oob_score': False, 'n_estimators': 35}
52    0.952464  0.00917326   0.49217   0.140354  0.0600024  {'class_weight': None, 'criterion': 'gini', 'oob_score': True, 'n_estimators': 45}
53    0.952464  0.00917326   0.49217   0.140354  0.0544893  {'class_weight': None, 'criterion': 'gini', 'oob_score': False, 'n_estimators': 45}
54    0.951343  0.00894095   0.489933  0.139774  0.0715628  {'class_weight': None, 'criterion': 'gini', 'oob_score': True, 'n_estimators': 50}
55    0.951343  0.00894095   0.489933  0.139774  0.0638719  {'class_weight': None, 'criterion': 'gini', 'oob_score': False, 'n_estimators': 50}
56    0.952464  0.00917326   0.485459  0.142819  0.0737503  {'class_weight': None, 'criterion': 'gini', 'oob_score': True, 'n_estimators': 55}
57    0.952464  0.00917326   0.485459  0.142819  0.0661024  {'class_weight': None, 'criterion': 'gini', 'oob_score': False, 'n_estimators': 55}
58    0.952464  0.00917326   0.489933  0.137804  0.132506   {'class_weight': None, 'criterion': 'gini', 'oob_score': True, 'n_estimators': 100}
59    0.952464  0.00917326   0.489933  0.137804  0.119919   {'class_weight': None, 'criterion': 'gini', 'oob_score': False, 'n_estimators': 100}
60    0.949666  0.00951709   0.485459  0.141503  0.0473779  {'class_weight': None, 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 30}
61    0.949666  0.00951709   0.485459  0.141503  0.0438493  {'class_weight': None, 'criterion': 'entropy', 'oob_score': False, 'n_estimators': 30}
62    0.951905  0.00923802   0.480984  0.140996  0.0546768  {'class_weight': None, 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 35}
63    0.951905  0.00923802   0.480984  0.140996  0.0495362  {'class_weight': None, 'criterion': 'entropy', 'oob_score': False, 'n_estimators': 35}
64    0.953024  0.00989268   0.480984  0.14143   0.070271   {'class_weight': None, 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 45}
65    0.953024  0.00989268   0.480984  0.14143   0.0636919  {'class_weight': None, 'criterion': 'entropy', 'oob_score': False, 'n_estimators': 45}
66    0.951904  0.00957987   0.480984  0.142214  0.0831263  {'class_weight': None, 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 50}
67    0.951904  0.00957987   0.480984  0.142214  0.0691881  {'class_weight': None, 'criterion': 'entropy', 'oob_score': False, 'n_estimators': 50}
68    0.953024  0.00989268   0.478747  0.141644  0.0849281  {'class_weight': None, 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 55}
69    0.953024  0.00989268   0.478747  0.141644  0.0780487  {'class_weight': None, 'criterion': 'entropy', 'oob_score': False, 'n_estimators': 55}
70    0.952464  0.00917326   0.480984  0.14143   0.144852   {'class_weight': None, 'criterion': 'entropy', 'oob_score': True, 'n_estimators': 100}
71    0.952464  0.00917326   0.480984  0.14143   0.123204   {'class_weight': None, 'criterion': 'entropy', 'oob_score': False, 'n_estimators': 100}

Accuracy on training set of the best model 0.941834451901566
Accuracy on testing set of the best model 0.7574750830564784
Confusion matrix for test set of the best model:
 [[212  26]
 [ 47  16]]

True Negatives:  212
False Positives:  26
False Negatives:  47
True Positives:  16

Parameters of the best model:  {'class_weight': None, 'criterion': 'gini', 'oob_score': True,
'n_estimators': 35}
```

Adaboost Model:

```
      trainAvg    trainSD    testAvg    testSD     fitTime  parameters
--  ----------  ---------  ---------  --------  ---------  ----------------------------------------
 0    0.807051  0.0171      0.691275  0.120837  0.0461563  {'n_estimators': 35, 'learning_rate': 1}
 1    0.812641  0.0195251   0.659955  0.156459  0.0716414  {'n_estimators': 45, 'learning_rate': 1}
 2    0.814874  0.0207593   0.657718  0.165347  0.0690173  {'n_estimators': 50, 'learning_rate': 1}
 3    0.815993  0.0220714   0.651007  0.145875  0.0482992  {'n_estimators': 55, 'learning_rate': 1}
 4    0.818235  0.0185171   0.639821  0.147932  0.0741716  {'n_estimators': 100, 'learning_rate': 1}
 5    0.833894  0.0161682   0.61745   0.147478  0.139151   {'n_estimators': 200, 'learning_rate': 1}

Accuracy on training set of the best model 0.8076062639821029
Accuracy on testing set of the best model 0.8106312292358804
Confusion matrix for test set of the best model:
 [[225  13]
 [ 44  19]]

True Negatives:  225
False Positives:  13
False Negatives:  44
True Positives:  19
Parameters of the best model:  {'n_estimators': 35, 'learning_rate': 1}
```

##

Task 2:

In task 2, I implemented and experimented with different hyper-parameters using grid search 5 different models: NN, KNN, LR, NB, and DT. I also constructed two ensemble classifiers from these 5 individual models: one with unweighted voting and the other with weighted voting based on the individual models' accuracies.

The neural net’s accuracy on training and testing data was 74% and 79% respectively. It performed very well in predicting that a person doesn’t donate blood – the true negative rate was 100%. However, the true positive rate was 0%, which is terrible. I also noticed that any little change in one hyper-parameter could throw off the neural net model. Based on this, I’m not sure if this neural net is appropriate for making predictions on this data…
The logistic regression model performed better. The training and testing accuracies were 77% and 81% respectively. The true negative and positive rates were 97% and 21% respectively.
The k-nearest neighbor model had a 80% accuracy on both training and testing data. The true negative and positive rate was 97% and 10% respectively – worse than logistic regression.
The Naive Bayes model had 70% and 80% accuracy on training and testing data respectively. The true negative and positive rate was 84% and 62% respectively – best true positive rate result than any of the individual models.
The decision tee model had 77% and 80% accuracy on training and testing data respectively. The true negative and positive rate was 96% and 19% respectively.
The ensemble classifier constructed from these 5 individual models using unweighted majority voting was implemented with Sklearn’s ensemble.VotingClassifier. It had 77% and 80% accuracy on training and testing data respectively. The true negative and positive rate was 100% and 2% respectively.
The weighted version of this ensemble classifier used the individual models accuracies to assign weights and had 77% and 81% accuracy on training and testing data respectively. The true negative and positive rate was 97% and 21% respectively – the weighted ensemble classifier performed better than the unweighted one.

The experiments with different hyper-parameters and best models are reported below.

Neural Net Model:

```
      trainAvg      trainSD    testAvg      testSD    fitTime  parameters
--  ----------  -----------  ---------  ----------  ---------
--------------------------------------------------------------------------------------------------------------------------------------------
 0    0.701987  0.0929184     0.693512  0.120686     0.760431  {'early_stopping': False, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver':
'adam', 'activation': 'relu', 'momentum': 0.9}
 1    0.760075  0.018045      0.682327  0.119445     0.595182  {'early_stopping': False, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver':
'lbfgs', 'activation': 'relu', 'momentum': 0.9}
 2    0.677932  0.152632      0.612975  0.192441     0.747497  {'early_stopping': False, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver':
'adam', 'activation': 'relu', 'momentum': 0.9}
 3    0.754494  0.0143051     0.682327  0.119445     0.464085  {'early_stopping': False, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver':
'lbfgs', 'activation': 'relu', 'momentum': 0.9}
 4    0.691326  0.0564617     0.697987  0.212528     1.07037   {'early_stopping': False, 'hidden_layer_sizes': (200,), 'learning_rate': 'constant', 'solver':
'adam', 'activation': 'relu', 'momentum': 0.9}
 5    0.749991  0.00951033    0.742729  0.00140825   0.607758  {'early_stopping': False, 'hidden_layer_sizes': (200,), 'learning_rate': 'constant', 'solver':
'lbfgs', 'activation': 'relu', 'momentum': 0.9}
```

```
 6    0.667338  0.148205      0.621924  0.187166     0.862667  {'early_stopping': False, 'hidden_layer_sizes': (200,), 'learning_rate': 'adaptive', 'solver':
'adam', 'activation': 'relu', 'momentum': 0.9}
 7    0.759531  0.0279669     0.686801  0.110534     1.53948   {'early_stopping': False, 'hidden_layer_sizes': (200,), 'learning_rate': 'adaptive', 'solver':
'lbfgs', 'activation': 'relu', 'momentum': 0.9}
 8    0.742729  0.000352649   0.742729  0.00140825   0.305771  {'early_stopping': True, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver':
'adam', 'activation': 'relu', 'momentum': 0.9}
 9    0.755027  0.0129343     0.682327  0.113963     0.86295   {'early_stopping': True, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver':
'lbfgs', 'activation': 'relu', 'momentum': 0.9}
10    0.739377  0.00656784    0.742729  0.00140825   0.277853  {'early_stopping': True, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver':
'adam', 'activation': 'relu', 'momentum': 0.9}
11    0.755608  0.0204406     0.691275  0.0961601    1.15537   {'early_stopping': True, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver':
'lbfgs', 'activation': 'relu', 'momentum': 0.9}
12    0.742729  0.000352649   0.742729  0.00140825   0.329615  {'early_stopping': True, 'hidden_layer_sizes': (200,), 'learning_rate': 'constant', 'solver':
'adam', 'activation': 'relu', 'momentum': 0.9}
13    0.744963  0.00462446    0.742729  0.00140825   0.844651  {'early_stopping': True, 'hidden_layer_sizes': (200,), 'learning_rate': 'constant', 'solver':
'lbfgs', 'activation': 'relu', 'momentum': 0.9}
14    0.742729  0.000352649   0.742729  0.00140825   0.361028  {'early_stopping': True, 'hidden_layer_sizes': (200,), 'learning_rate': 'adaptive', 'solver':
'adam', 'activation': 'relu', 'momentum': 0.9}
15    0.769031  0.025304      0.689038  0.106079     1.5848    {'early_stopping': True, 'hidden_layer_sizes': (200,), 'learning_rate': 'adaptive', 'solver':
'lbfgs', 'activation': 'relu', 'momentum': 0.9}
16    0.775734  0.0298016     0.677852  0.122867     3.45001   {'early_stopping': False, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver':
'adam', 'activation': 'tanh', 'momentum': 0.9}
17    0.770713  0.0369647     0.671141  0.141722     2.60969   {'early_stopping': False, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver':
'lbfgs', 'activation': 'tanh', 'momentum': 0.9}
18    0.769041  0.032444      0.673378  0.137267     2.86607   {'early_stopping': False, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver':
'adam', 'activation': 'tanh', 'momentum': 0.9}
19    0.792523  0.0252312     0.662192  0.159741     3.08617   {'early_stopping': False, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver':
'lbfgs', 'activation': 'tanh', 'momentum': 0.9}
20    0.779098  0.0227967     0.626398  0.145709     5.27471   {'early_stopping': False, 'hidden_layer_sizes': (200,), 'learning_rate': 'constant', 'solver':
'adam', 'activation': 'tanh', 'momentum': 0.9}
21    0.770693  0.0186322     0.66443   0.149584     4.44265   {'early_stopping': False, 'hidden_layer_sizes': (200,), 'learning_rate': 'constant', 'solver':
'lbfgs', 'activation': 'tanh', 'momentum': 0.9}
22    0.775176  0.0296431     0.668904  0.140678     3.55206   {'early_stopping': False, 'hidden_layer_sizes': (200,), 'learning_rate': 'adaptive', 'solver':
'adam', 'activation': 'tanh', 'momentum': 0.9}
23    0.778555  0.033393      0.659955  0.163999     5.11669   {'early_stopping': False, 'hidden_layer_sizes': (200,), 'learning_rate': 'adaptive', 'solver':
'lbfgs', 'activation': 'tanh', 'momentum': 0.9}
24    0.730997  0.0206232     0.657718  0.116548     0.3588    {'early_stopping': True, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver':
'adam', 'activation': 'tanh', 'momentum': 0.9}
25    0.776862  0.0357687     0.657718  0.162944     2.06182   {'early_stopping': True, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver':
'lbfgs', 'activation': 'tanh', 'momentum': 0.9}
26    0.742729  0.000352649   0.742729  0.00140825   0.342828  {'early_stopping': True, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver':
'adam', 'activation': 'tanh', 'momentum': 0.9}
27    0.779654  0.0339567     0.659955  0.158491     2.5325    {'early_stopping': True, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver':
'lbfgs', 'activation': 'tanh', 'momentum': 0.9}
28    0.736025  0.0132678     0.731544  0.0230465    0.49719   {'early_stopping': True, 'hidden_layer_sizes': (200,), 'learning_rate': 'constant', 'solver':
'adam', 'activation': 'tanh', 'momentum': 0.9}
29    0.782453  0.0246786     0.666667  0.145131     5.41778   {'early_stopping': True, 'hidden_layer_sizes': (200,), 'learning_rate': 'constant', 'solver':
'lbfgs', 'activation': 'tanh', 'momentum': 0.9}
30    0.742729  0.000352649   0.742729  0.00140825   0.499546  {'early_stopping': True, 'hidden_layer_sizes': (200,), 'learning_rate': 'adaptive', 'solver':
'adam', 'activation': 'tanh', 'momentum': 0.9}
31    0.775743  0.0296768     0.657718  0.13663      3.45556   {'early_stopping': True, 'hidden_layer_sizes': (200,), 'learning_rate': 'adaptive', 'solver':
'lbfgs', 'activation': 'tanh', 'momentum': 0.9}

Accuracy on training set of the best model 0.7427293064876958
Accuracy on testing set of the best model 0.7906976744186046
Confusion matrix for test set of the best model:
 [[238   0]
 [ 63   0]]

True Negatives:  238
False Positives:  0
False Negatives:  63
True Positives:  0
Parameters of the best model:  {'early_stopping': False, 'hidden_layer_sizes': (200,),
'learning_rate': 'constant', 'solver': 'lbfgs', 'activation': 'relu', 'momentum': 0.9}
```

Logistic Regression Model:

```
      trainAvg    trainSD    testAvg    testSD     fitTime  parameters
--  ----------  ---------  ---------  --------  ----------  --------------------------------------------------------------------------------------------
 0    0.775735  0.0226927   0.691275  0.101624  0.00607843  {'solver': 'liblinear', 'fit_intercept': False, 'max_iter': 200, 'C': 1, 'penalty': 'l2'}
 1    0.775735  0.0226927   0.691275  0.101624  0.0160131   {'solver': 'liblinear', 'fit_intercept': False, 'max_iter': 200, 'C': 1, 'penalty': 'l1'}
 2    0.775735  0.0226927   0.691275  0.101624  0.00654225  {'solver': 'liblinear', 'fit_intercept': False, 'max_iter': 400, 'C': 1, 'penalty': 'l2'}
 3    0.775735  0.0226927   0.691275  0.101624  0.00578866  {'solver': 'liblinear', 'fit_intercept': False, 'max_iter': 400, 'C': 1, 'penalty': 'l1'}
 4    0.769031  0.0257926   0.689038  0.10061   0.00393224  {'solver': 'liblinear', 'fit_intercept': True, 'max_iter': 200, 'C': 1, 'penalty': 'l2'}
 5    0.770152  0.0250839   0.689038  0.10061   0.00646853  {'solver': 'liblinear', 'fit_intercept': True, 'max_iter': 200, 'C': 1, 'penalty': 'l1'}
 6    0.769031  0.0257926   0.689038  0.10061   0.00525131  {'solver': 'liblinear', 'fit_intercept': True, 'max_iter': 400, 'C': 1, 'penalty': 'l2'}
 7    0.770152  0.0250839   0.689038  0.10061   0.0102907   {'solver': 'liblinear', 'fit_intercept': True, 'max_iter': 400, 'C': 1, 'penalty': 'l1'}
 8    0.775735  0.0226927   0.691275  0.101624  0.00481076  {'solver': 'liblinear', 'fit_intercept': False, 'max_iter': 200, 'C': 1.5, 'penalty': 'l2'}
 9    0.776294  0.0229448   0.691275  0.101624  0.00750132  {'solver': 'liblinear', 'fit_intercept': False, 'max_iter': 200, 'C': 1.5, 'penalty': 'l1'}
10    0.775735  0.0226927   0.691275  0.101624  0.00390973  {'solver': 'liblinear', 'fit_intercept': False, 'max_iter': 400, 'C': 1.5, 'penalty': 'l2'}
11    0.775735  0.0226927   0.691275  0.101624  0.01255     {'solver': 'liblinear', 'fit_intercept': False, 'max_iter': 400, 'C': 1.5, 'penalty': 'l1'}
12    0.769031  0.0257926   0.689038  0.10061   0.00509901  {'solver': 'liblinear', 'fit_intercept': True, 'max_iter': 200, 'C': 1.5, 'penalty': 'l2'}
13    0.769591  0.0254161   0.689038  0.10061   0.0109598   {'solver': 'liblinear', 'fit_intercept': True, 'max_iter': 200, 'C': 1.5, 'penalty': 'l1'}
14    0.769031  0.0257926   0.689038  0.10061   0.00540566  {'solver': 'liblinear', 'fit_intercept': True, 'max_iter': 400, 'C': 1.5, 'penalty': 'l2'}
15    0.769591  0.0254161   0.689038  0.10061   0.0125201   {'solver': 'liblinear', 'fit_intercept': True, 'max_iter': 400, 'C': 1.5, 'penalty': 'l1'}
16    0.775735  0.0226927   0.691275  0.101624  0.00691018  {'solver': 'liblinear', 'fit_intercept': False, 'max_iter': 200, 'C': 2, 'penalty': 'l2'}
17    0.775735  0.0226927   0.691275  0.101624  0.0134206   {'solver': 'liblinear', 'fit_intercept': False, 'max_iter': 200, 'C': 2, 'penalty': 'l1'}
18    0.775735  0.0226927   0.691275  0.101624  0.00354247  {'solver': 'liblinear', 'fit_intercept': False, 'max_iter': 400, 'C': 2, 'penalty': 'l2'}
19    0.775735  0.0226927   0.691275  0.101624  0.0157171   {'solver': 'liblinear', 'fit_intercept': False, 'max_iter': 400, 'C': 2, 'penalty': 'l1'}
20    0.769031  0.0257926   0.689038  0.10061   0.00476537  {'solver': 'liblinear', 'fit_intercept': True, 'max_iter': 200, 'C': 2, 'penalty': 'l2'}
21    0.769591  0.0254161   0.689038  0.10061   0.0119183   {'solver': 'liblinear', 'fit_intercept': True, 'max_iter': 200, 'C': 2, 'penalty': 'l1'}
22    0.769031  0.0257926   0.689038  0.10061   0.00430937  {'solver': 'liblinear', 'fit_intercept': True, 'max_iter': 400, 'C': 2, 'penalty': 'l2'}
23    0.769591  0.0254161   0.689038  0.10061   0.0108385   {'solver': 'liblinear', 'fit_intercept': True, 'max_iter': 400, 'C': 2, 'penalty': 'l1'}
```

```
Accuracy on training set of the best model 0.7695749440715883
Accuracy on testing set of the best model 0.813953488372093
Confusion matrix for test set of the best model:
 [[232   6]
 [ 50  13]]
```

```
True Negatives:  232
False Positives:  6
False Negatives:  50
True Positives:  13
Parameters of the best model:  {'solver': 'liblinear', 'fit_intercept': False, 'max_iter': 200, 'C':
1, 'penalty': 'l2'}
```

KNN Model:

```
     trainAvg     trainSD    testAvg    testSD     fitTime  parameters
--  ----------  ----------  ---------  --------  ----------  ------------------
 0    0.915552  0.0199692    0.469799  0.156158  0.00345845  {'n_neighbors': 1}
 1    0.842841  0.00271259   0.58613   0.135427  0.00404477  {'n_neighbors': 2}
 2    0.829429  0.00933157   0.532438  0.151106  0.00355911  {'n_neighbors': 3}
 3    0.810402  0.00649594   0.610738  0.132784  0.00287805  {'n_neighbors': 4}
 4    0.802576  0.00362357   0.574944  0.151764  0.0018003   {'n_neighbors': 5}

Accuracy on training set of the best model 0.7986577181208053
Accuracy on testing set of the best model 0.7906976744186046
Confusion matrix for test set of the best model:
 [[232   6]
 [ 57   6]]

True Negatives:  232
False Positives:  6
False Negatives:  57
True Positives:  6
Parameters of the best model:  {'n_neighbors': 4}
```

Naive Bayes Model:

```
     trainAvg    trainSD    testAvg    testSD     fitTime  parameters
--  ----------  ---------  ---------  --------  ----------  ---------------------------------
 0    0.659395  0.0408111   0.624161  0.20432   0.00295916  {'fit_prior': False, 'alpha': 1.0}
 1    0.686767  0.036875    0.668904  0.213613  0.00367055  {'fit_prior': True, 'alpha': 1.0}
 2    0.659395  0.0408111   0.624161  0.20432   0.003157    {'fit_prior': False, 'alpha': 1.5}
 3    0.686207  0.0369844   0.668904  0.213613  0.00223784  {'fit_prior': True, 'alpha': 1.5}

Accuracy on training set of the best model 0.6957494407158836
Accuracy on testing set of the best model 0.7973421926910299
```

```
Confusion matrix for test set of the best model:
 [[201  37]
 [ 24  39]]

True Negatives:  201
False Positives:  37
False Negatives:  24
True Positives:  39
Parameters of the best model:  {'fit_prior': True, 'alpha': 1.0}
```

Decision Tree Model:

```
     trainAvg     trainSD    testAvg     testSD     fitTime  parameters
--  ----------  ----------  ---------  ---------  ----------  -------------------------------------------------------------------------------------
 0    0.953584  0.00959807   0.480984  0.14516    0.00287395  {'criterion': 'gini', 'max_depth': None, 'splitter': 'best', 'max_features': None}
 1    0.953584  0.00959807   0.496644  0.127279   0.00274749  {'criterion': 'gini', 'max_depth': None, 'splitter': 'random', 'max_features': None}
 2    0.953584  0.00959807   0.438479  0.138545   0.00266838  {'criterion': 'gini', 'max_depth': None, 'splitter': 'best', 'max_features': 3}
 3    0.953584  0.00959807   0.436242  0.154151   0.00228758  {'criterion': 'gini', 'max_depth': None, 'splitter': 'random', 'max_features': 3}
 4    0.827185  0.0174426    0.599553  0.102248   0.00318975  {'criterion': 'gini', 'max_depth': 5, 'splitter': 'best', 'max_features': None}
 5    0.77405   0.0163224    0.637584  0.130757   0.00287609  {'criterion': 'gini', 'max_depth': 5, 'splitter': 'random', 'max_features': None}
 6    0.825511  0.0171523    0.595078  0.0957274  0.0030817   {'criterion': 'gini', 'max_depth': 5, 'splitter': 'best', 'max_features': 3}
 7    0.768473  0.0252328    0.671141  0.0921414  0.00287347  {'criterion': 'gini', 'max_depth': 5, 'splitter': 'random', 'max_features': 3}
 8    0.927848  0.00839673   0.469799  0.149746   0.00299788  {'criterion': 'gini', 'max_depth': 10, 'splitter': 'best', 'max_features': None}
 9    0.878655  0.0313526    0.548098  0.131167   0.00186682  {'criterion': 'gini', 'max_depth': 10, 'splitter': 'random', 'max_features': None}
10    0.924497  0.0133382    0.49217   0.138173   0.00201397  {'criterion': 'gini', 'max_depth': 10, 'splitter': 'best', 'max_features': 3}
11    0.867454  0.0318022    0.494407  0.142492   0.00181704  {'criterion': 'gini', 'max_depth': 10, 'splitter': 'random', 'max_features': 3}
12    0.953584  0.00959807   0.438479  0.158606   0.00269852  {'criterion': 'entropy', 'max_depth': None, 'splitter': 'best', 'max_features': None}
13    0.953584  0.00959807   0.46085   0.152621   0.00320754  {'criterion': 'entropy', 'max_depth': None, 'splitter': 'random', 'max_features': None}
14    0.953584  0.00959807   0.436242  0.161007   0.00355492  {'criterion': 'entropy', 'max_depth': None, 'splitter': 'best', 'max_features': 3}
15    0.953584  0.00959807   0.465324  0.161597   0.00316033  {'criterion': 'entropy', 'max_depth': None, 'splitter': 'random', 'max_features': 3}
16    0.810976  0.0232146    0.644295  0.116851   0.0029923   {'criterion': 'entropy', 'max_depth': 5, 'splitter': 'best', 'max_features': None}
17    0.784126  0.0211833    0.673378  0.0870484  0.001828    {'criterion': 'entropy', 'max_depth': 5, 'splitter': 'random', 'max_features': None}
18    0.806496  0.0172932    0.673378  0.067519   0.00200353  {'criterion': 'entropy', 'max_depth': 5, 'splitter': 'best', 'max_features': 3}
19    0.775744  0.0184626    0.706935  0.0649282  0.00181274  {'criterion': 'entropy', 'max_depth': 5, 'splitter': 'random', 'max_features': 3}
20    0.906042  0.00945347   0.478747  0.151222   0.00227804  {'criterion': 'entropy', 'max_depth': 10, 'splitter': 'best', 'max_features': None}
21    0.890927  0.0159168    0.474273  0.150686   0.00194354  {'criterion': 'entropy', 'max_depth': 10, 'splitter': 'random', 'max_features': None}
22    0.899332  0.0114507    0.514541  0.134104   0.00227675  {'criterion': 'entropy', 'max_depth': 10, 'splitter': 'best', 'max_features': 3}
23    0.865773  0.0138032    0.498881  0.142      0.00194383  {'criterion': 'entropy', 'max_depth': 10, 'splitter': 'random', 'max_features': 3}
```

```
Accuracy on training set of the best model 0.7718120805369127
Accuracy on testing set of the best model 0.8006644518272426
Confusion matrix for test set of the best model:
 [[229   9]
 [ 51  12]]

True Negatives:  229
```

```
False Positives:  9
False Negatives:  51
True Positives:  12
Parameters of the best model:  {'criterion': 'entropy', 'max_depth': 5, 'splitter': 'random',
'max_features': 3}
```

Ensemble model: unweighted majority voting over 5 models

```
Accuracy on training set of the ensemble model 0.7695749440715883
Accuracy on testing set of the ensemble model 0.7940199335548173
Confusion matrix for test set:
 [[238   0]
 [ 62   1]]

True Negatives:  238
False Positives:  0
False Negatives:  62
True Positives:  1
```

Ensemble model: weighted majority voting over 5 models (weights proportional to accuracy)

```
Accuracy on training set of the ensemble model 0.7718120805369127
Accuracy on testing set of the ensemble model 0.813953488372093
Confusion matrix for test set:
 [[232   6]
 [ 50  13]]

True Negatives:  232
False Positives:  6
False Negatives:  50
True Positives:  13
```

##

Task 3:

In task 3, I coded two ensemble classifiers based on the 7 different models from task 1 and task 2: one with unweighted voting and the other with weighted voting based on the individual models' accuracies.
The unweighted ensemble classifier had 82% accuracy on both training and testing data and true negative and positive rate of 97% and 25% respectively.
The weighted ensemble classifier used the accuracies of the 7 models as weights and had 82% and 81% accuracy on training and testing data respectively and a true negative and positive rate of 98% and 17% respectively.

Ensemble model: unweighted majority voting over 7 models

```
Accuracy on training set of the ensemble model 0.8165548098434005
Accuracy on testing set of the ensemble model 0.8172757475083057
Confusion matrix for test set:
 [[230   8]
 [ 47  16]]

True Negatives:  230
False Positives:  8
False Negatives:  47
True Positives:  16
```

Ensemble model: weighted majority voting over 7 models (weights proportional to accuracy)

```
Accuracy on training set of the ensemble model 0.8187919463087249
Accuracy on testing set of the ensemble model 0.8106312292358804
Confusion matrix for test set:
 [[233   5]
 [ 52  11]]

True Negatives:  233
False Positives:  5
False Negatives:  52
True Positives:  11
```

In summary, all models had high training and testing accuracies. The best accuracy on testing data was 82% and was achieved by the unweighted ensemble classifier comprised of all 7 models. The best testing accuracy using an individual model and not an ensemble was 81% achieved by AdaBoost and logistic regression. All models performed quite well on predicting negative outcomes – not donating blood. The true negative rate was high for all individual and ensemble classifiers – above 84%. However, all models did not perform too well at predicting positive outcomes – donating blood. The best true positive rate was 62% and was achieved by the Naive Bayes model. Wheather this is an issue or not depends on what is more important to predict correctly – donating or not donating blood? If correct predictions of not donating blood are of interest and more important than correct predictions of donating blood, these models are very suitable.