Q1.

Logistic Regression:

Logistic Regression is a supervised machine learning algorithm that can be used to model the probability of a certain class or event. The logistic function always produces an S-shaped curve regardless of the value of X. The predictions are made by applying the sigmoid function. Logistic regression does not really have any critical hyperparameters to tune. Regularization (penalty) can sometimes be helpful. ometimes, you can see useful differences in performance or convergence with different solvers (solver). The C parameter controls the penality strength, which can also be effective.

In this example we have used the following hyperparameters to tune and form the grid

solver = ['newton-cg', 'lbfgs', 'liblinear'], penalty = ['l2'], c_values = [100, 10, 1.0, 0.1, 0.01]

grid=dict(solver=solvers,penalty=penalty,C=c_values)

Ridge Classifier:

This classifier first converts the target values into {-1, 1} and then treats the problem as a regression task (multi-output regression in the multiclass case). Ridge regression is a penalized linear regression model for predicting a numerical value. it can be very effective when applied to classification. Perhaps the most important parameter to tune is the regularization strength (alpha). A good starting point might be values in the range [0.1 to 1.0]

In this example we have used the following hyperparameters to tune and form the grid:

alpha = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0](grid = dict(alpha=alpha))

KNN classifier:

KNeighborsClassifier implements learning based on the nearest neighbors of each query point, where is an integer value specified by the user.

Test values between at least 1 and 21, perhaps just the odd numbers. It may also be interesting to test different distance metrics (metric) for choosing the composition of the neighborhood. It is also useful to test the contribution of members of the neighborhood via different weightings (weights). In this example we have used the following hyperparameters to tune and form the grid:

n_neighbors = range(1, 21, 2), weights = ['uniform', 'distance']

metric = ['euclidean', 'manhattan', 'minkowski']

grid = dict(n_neighbors=n_neighbors,weights=weights,metric=metric)

Random Forests:

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

The most important parameter is the number of random features to sample at each split point (max_features). Another important parameter for random forest is the number of trees (n_estimators). Ideally, this should be increased until no further improvement is seen in the model. Good values might be a log scale from 10 to 1,000.

Q2.

roc_auc_score and f1_score are the performance metrics used to select the best model overall.

The roc_auc_score is the area under the ROC curve, which is a chart that visualizes the tradeoff between true positive rate (TPR) and false-positive rate (FPR). This function requires the true binary value and the target scores, which can either be probability estimates of the positive class, confidence values, or binary decisions. Since we ultimately care about ranking predictions and not about outputting well-calibrated probabilities, roc_auc_score is a great metric to evaluate our model on.

The F1 score can be looked at as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. Precision and recall have almost equal contribution to the F1 score as it combines the precision and recall of a classifier into a single metric by taking their harmonic mean. The formula for the F1 score is:

F1 = 2 * (precision * recall) / (precision + recall)

F1-score is a better metric when there are imbalanced classes. Since in most real-life classification problems, there is an imbalanced class distribution, F1-score is a great metric to evaluate our model on.

Q3.

Best hyperparameter settings of logistic regression:

C: 100, penalty: 'l2', solver: 'newton-cg'

Best hyperparameter settings of ridge classifier:

alpha: 0.1

Best hyperparameter setting of KNN classifier:

metric: 'manhattan', n_neighbors: 19, weights: 'distance'

Best hyperparameter of Random Forest Classifier:

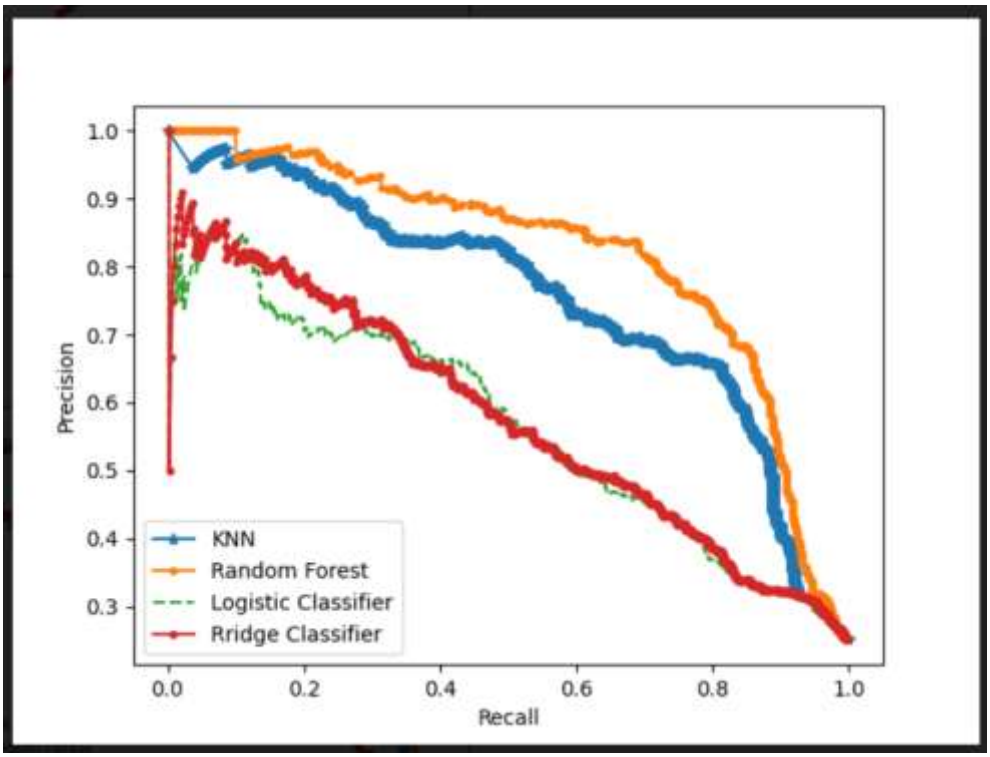
max_features: 'sqrt', n_estimators: 1000

Q4.

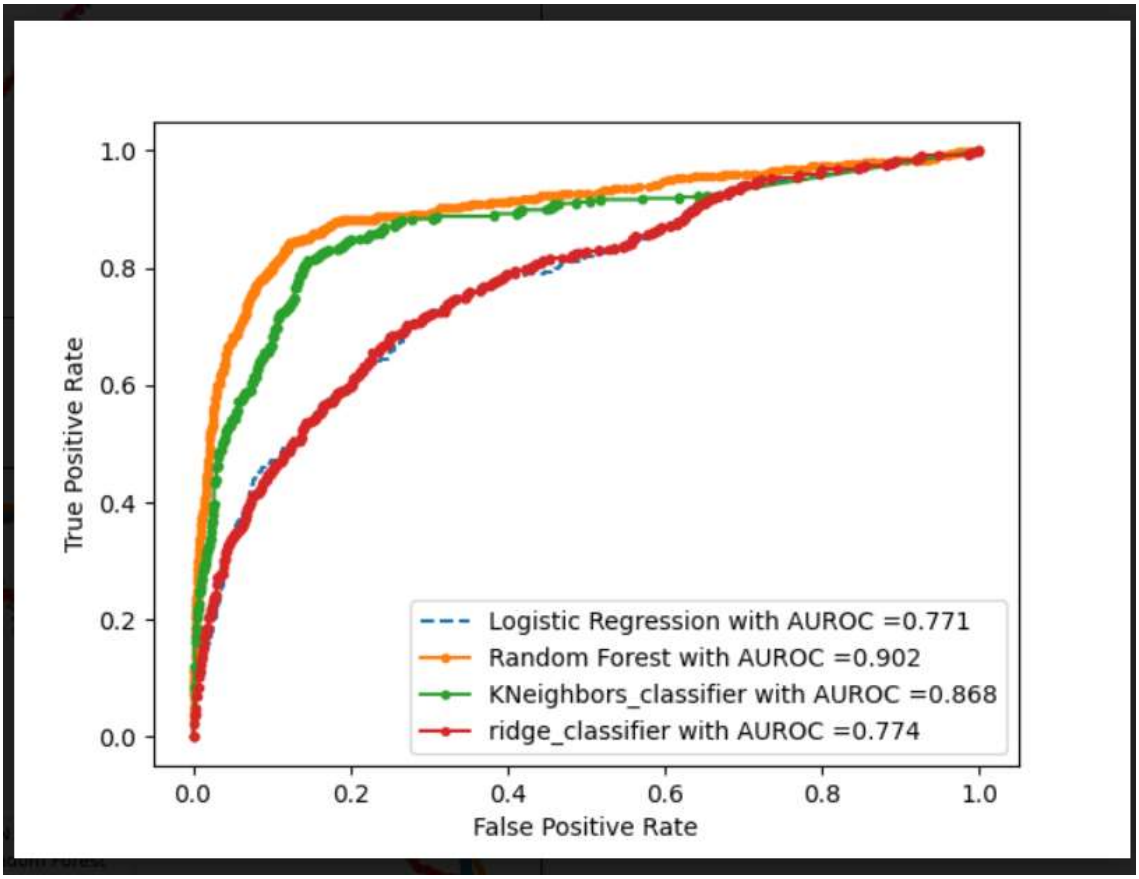
	Mean	SD
Logistic Regression	0.795502	0.020693
Ridge Classifier	0.795502	0.017998
KNN Classifier	0.856206	0.017794
Random Forest Classifier	0.886673	0.012949

Random Forest Classifier is the model selected to generate the final model.

Q5.



Q6.



Q7:

Acknowledgement section:

https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

https://scikit-learn.org/stable/tutorial/statistical_inference/model_selection.html

https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc_crossval.html

https://towardsdatascience.com/complete-guide-to-pythons-cross-validation-with-examples-a9676b5cac12

https://towardsdatascience.com/complete-guide-to-pythons-cross-validation-with-examples-a9676b5cac12

https://machinelearningmastery.com/k-fold-cross-validation/

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/

Program specification:
Python version: 3.10
Running in the command Line of windows 10 assuming the data files are in the same directory:
A3_Class.py A3_Traindata.tsv A3_Testdata.tsv
all of grid-search CV in the code is commented out to run faster, they all run as supposed to and can be uncommented to run.
Libraries used: NumPy, pandas, matplotlib, scikit-learn, sys, re