Exercise 1 - Support Vector Machine

The decision function for SVMs is:
f(x_{test}) = Σ{N},{i=1}[α_{i} * y_{i} * < x_{i}, x_{test} > + b]
In our function, we use the summation over all training samples **(N)**, of the product of:
1) The Lagrange multipliers for each training sample – ***α_{i}*** (0 <= α_{i} <= C, where C = 10 in our case)
2) The class labels of the training samples – ***y_{i}***
3) The dot product between each training sample/vector and the test sample/vector – ***x_{i}, x_{test}*** respectively. The dot product is a measure of similarity between the training and test sample/vector. This means, similar training and test samples will have a large dot product, while the dot product will be small or even negative if the samples are dissimilar.
After that, each dot product is weighted by its corresponding Lagrange multiplier. The class labels will provide us with negative or positive contributions if -1 or 1 respectively.

We also add the bias term ***b,*** which is a scalar and can be moved outside the summation for simplicity. The bias term will adjust the threshold at which the decision boundary will start changing classification (from class 1 to class 2 for example).

To compute the bias term, we will use the decision function and all of the samples that do not have α = 0, since they are not support vectors and do not contribute to the decision boundary.

So for the sample, we have x_i = [X_1, X2, X3]:

x_1 = [1, 0, 1], α1 = 1, y1 = -1
x_2 = [0, -1, 0], α2 = 1, y2 = 1
x_3 = [1, 1, -1], α3 = 10 (margin violation/misclassification since C = 10), y3 = -1

We will choose x3 as our test sample.
So, b = y3 - Σ{3},{i=1}[α_{i} * y_{i} * < x_{i}, x_{3} >]

- <x1,x3> = [1, 0, 1] * [1, 1, -1] = (1*1) + (0*1) + (1*-1) = 0
- <x2,x3> = [0, -1, 0] * [1, 1, -1] = (0*1) + (1*-1) + (0*-1) = -1
- <x3,x3> = [1, 1, -1] * [1, 1, -1] = (1*1) + (1*1) + (-1*-1) = 3

For i=1: α_{1} * y_{1} * < x_{1}, x_{3} > = 1 * (-1) * 0 = 0
For i=2: α_{2} * y_{2} * < x_{2}, x_{3} > = 1 * 1 * (-1) = -1
For i=3: α_{3} * y_{3} * < x_{3}, x_{3} > = 10 * (-1) * 3 = -30
So the summation of the above is -31, thus: b = -1 - (-31) → **b = 30**

Let us do the same, now choosing x1 and x2 as test samples.

With x1:

- <x1,x1> = [1, 0, 1] * [1,0,1] = (1*1) + (0*0) + (1*1) = 2
- <x2,x1> = [0, -1, 0] * [1,0,1] = (0*1) + (-1*0) + (0*1) = 0
- <x3,x1> = [1, 1, -1] * [1,0,1] = (1*1) + (1*0) + (-1*1) = 0

For i=1: α_{1} * y_{1} * < x_{1}, x_{1} > = 1 * (-1) * 2 = -2
For i=2: α_{2} * y_{2} * < x_{2}, x_{1} > = 1 * 1 * (0) = 0
For i=3: α_{3} * y_{3} * < x_{3}, x_{1} > = 10 * (-1) * 0 = 0

So the summation of the above is -2. b = y1 - (-2) = -1 - (-2) → **b = 1**

With x2:

- <x1,x2> = [1, 0, 1] * [0, -1, 0] = (1*0) + (0*-1) + (1*0) = 0
- <x2,x2> = [0, -1, 0] * [0, -1, 0] = (0*0) + (-1*-1) + (0*0) = 1
- <x3,x2> = [1, 1, -1] * [0, -1, 0] = (1*0) + (1*-1) + (1*0) = -1

For i=1: α_{1} * y_{1} * < x_{1}, x_{1} > = 1 * (-1) * 0 = 0
For i=2: α_{2} * y_{2} * < x_{2}, x_{1} > = 1 * 1 * (1) = 1
For i=3: α_{3} * y_{3} * < x_{3}, x_{1} > = 10 * (-1) * (-1) = 10

So the summation of the above is 11. b = y2 - (11) = 1 - (11) → **b = -10**

Each test sample gives us a different bias term ***b,*** which is something we do not want. In a properly trained SVM model, we would expect the b to be the same no matter what sample we use that has a Lagrange multiplier 0 < α < C. This means we probably did not derive correct Lagrange multipliers from the training process. This could also answer our second problem indirectly (*Show how the provided Lagrange multipliers cannot really be the solution to the proposed problem).*

Another thing we could if we knew ξi (slack variable) is evaluate complementary slackness, where we could check ***α_{i}* [y_{i}*(< x{i}, w > + b) -1 + ξ_{i}] = 0***
We can compute **w** as: ***w = Σ_{N},{i=1} [α_{i}*x_{i}*y_{i}]***

The Guassian kernel for two vectors x_{i} and x_{j}:
K(x, z) = exp(- (|| x - z ||^2) / 2(σ)^2)

We know that σ = 1, so the above becomes:
K(x, z) = exp(- (|| x - z ||^2) / 2)

So the decision function becomes:
f(x_{test}) = Σ{N},{i=1}[α_{i} * y_{i} * (K(x_{i}, x_{test})) + b]

To compute b:
b = yi - Σ{N},{i=1}[α_{i} * y_{i} * K(x_{i}, x_{test})]

Again, let us choose all 3 samples as support vectors (***x_{s}***)
This time, let's start with x1 and calculate K(x_{i}, x1)

- || x1 - x1 ||^2 = 0
- || x2 - x1 ||^2 = (0 - 1)^2 + (-1-0)^2 + (0-1)^2 = 1 + 1 + 1 = 3
- || x3 - x1 ||^2 = (1-1)^2 + (1-0)^2 + (-1-1)^2 = 5

K(x1,x1) = exp(0) = 1
K(x2,x1) = exp(-3/2) ~= 0.223
K(x3,x1) = exp(-5/2) ~= 0.082

For i=1: α_{1} * y_{1} * K(x1,x1) = 1 * (-1) * 1 = -1
For i=2: α_{2} * y_{2} * K(x2,x1) = 1 * 1 * 0.223 = 0.223
For i=3: α_{3} * y_{3} * K(x3,x1) = 10 * (-1) * (0.082) = -0.82
The summation is (-1) + 0.223 + (-0.82) = -1.597

b = -1 - (-1.597) → **b = 0.597**

For x2:

- || x1 - x2 ||^2 = (1-0)^2 + (0+1)^2 + (1-0)^2 = 3
- || x2 - x2 ||^2 = 0
- || x3 - x2 ||^2 = (1-0)^2 + (1+1)^2 + (-1-0)^2 = 6

K(x1,x2) = exp(-3/2) ~= 0.223
K(x2,x2) = exp(0) = 1
K(x3,x2) = exp(-6/2) ~= 0.045

For i=1: α_{1} * y_{1} * K(x1,x2) = 1 * (-1) * 0.223 = -0.223
For i=2: α_{2} * y_{2} * K(x2,x2) = 1 * 1 * 1 = 1
For i=3: α_{3} * y_{3} * K(x3,x2) = 10 * (-1) * 0.045 = -0.45
The summation is 0.327
b = 1 - 0.327 → **b = 0.673**

For x3:

- || x1 - x3 ||^2 = (1-1)^2 + (0-1)^2 + (1+1)^2 = 5
- || x2 - x3 ||^2 = (0-1)^2 + (-1-1)^2 + (0+1)^2 = 6
- || x3 - x2 ||^2 = 0

K(x1,x3) = exp(-5/2) ~= 0.082
K(x2,x3) = exp(-6/2) ~= 0.045
K(x3,x3) = exp(0) = 1

For i=1: α_{1} * y_{1} * K(x1,x3) = 1 * (-1) * 0.082 = -0.082
For i=2: α_{2} * y_{2} * K(x2,x3) = 1 * 1 * 0.045 = 0.045
For i=3: α_{3} * y_{3} * K(x3,x3) = 10 * (-1) * (1) = -10
The summation is -10.037

b = -1 - (-10.037) → **b = 9.037**

Again we notice different bias terms, but at least we have very small fluctuations between x1 and x2 samples (α = 1) compared to x3 where α = C = 10, which is as mentioned before a margin violation or misclassification. I suppose a generally good idea (we might want to exclude for α = C) to use an average of the bias terms if they do not have considerably high differences, and move on with that bias term. This might not be a good strategy for the first case, since we had higher variations of bias terms (1, -10, 30).

Exercise 2 - Causal-Based Feature Selection

Using the pseudo code, we will first create the forward and backward selection functions.

Forward selection:
The idea is that we start with no features and add them one by one.

First, we create an empty set to store any selected features and create a dynamic set that will contain all the remaining variables (R). While S changes, so will R. Then we will initiate an infinite while True loop, which will have a condition to break later on (if S does not change).
To do this, we create a set ***S_prev*** which will create a copy of our S set (selected variables) as it changes in each iteration. We create a plus infinite minimum p-value and create an empty variable ***V_star*** which will later contain any features that are under consideration to be included in our feature selection.
Now we start iterating through the R set (remaining variables) which contains indexes for the features, and for each, we calculate the p-value against the target variable, conditioning on S. We then check if the p-value is lower than our minimum p-value (starting with plus infinity, this will always be true for the first index) and if the condition is met, we set that p-value as our new minimum p-value and store said feature as our ***V_star*** (this is always reset at the beginning of a new main loop). If we get a feature, we remove it from our set R and then check if the p-value of that feature is lower than the threshold alpha. If this condition is met it means the feature is statistically significant and thus we add it to our selected feature set (S). At the end of the main loop, we check if the S generated is the same as the previous one, in which case we break the loop and return the selected feature indices set.

Backward selection:
The idea is now that we start with all the features and remove them one by one.

Now our initial set S is all of the features. Again the idea is that we iterate through each feature, and create in the same manner ***S_prev*** and ***V_star*** as previously, only this time ***S_prev*** should remove features while the loop runs (previously we started with an empty set and started filling in features).
Again, we iterate through each feature index contained in the S set, create a new set by removing the feature from the S set and then calculate the p-value for the current variable against the target variable, conditioning on S without the current variable this time. The same statistical checks follow, with a difference that this time our starting p-value is now a maximum p-value, initialized as minus infinite, so the first feature will always have a higher p-value. So, if the p-value for the current feature is higher than the maximum p-value, that p-value becomes the new maximum p-value, and said feature is taken into consideration, this time for exclusion. If that maximum p-value is higher than our threshold alpha, we remove it from our dynamic set S. At the end of

each loop again we check if ***S_prev*** is the same as the set S to break the loop and return the backward selection feature indices set.

5-Fold Cross Validation:

For this function, we will use a default of k_folds = 5. The function uses as input the features (X), the target variables (class – Y), the classifier (in our case SVM and RandomForest), the parameters for the classifier as a dictionary and the k-folds, which is set to 5. Using the ***StratifiedKFold*** from sklearn model selection, we initialize a StratifiedKFold object for 5-fold cross-validation (because stratification ensures each fold is a good representative of the whole), by also shuffling data before splitting into batches. This time we kept **random_state = 42** to problem solve later on, but it can be set to **None**, like the previous Assignment.
Now that we have our folds, we will iterate over each one and keep the training and validation (test) data, which we will then use to create our model using the selected classifier and parameters, train it then predict probabilities on the test data. Lastly, we calculate the ROC AUC score and keep it in our list of ROC AUC scores. After we have gone through each fold, it returns the mean ROC AUC score.

Now that we have set up our functions (including 2 functions to create a few plots), we can start the model selection and evaluation. Since the initial dataset D contains the target variable as the last column, we keep that as a separate variable ***target_var*** and also remove it from our variable ***features*** (containing all but the last column). We will split our data into 80% training and 20% test. Since forward and backward selection functions were set up to use all the features in the dataset, we will create a new one by concatenating the X and y training data into a new dataset ***D_train***. We will use this dataset to run forward and backward selection on all 3 alpha thresholds (0.05, 0.01, 0.005).

The results as seen in the ***results.out*** file are these:

- **For a = 0.05:**
  - Forward_Selection_Variables: [0, 24, 21, 23]
  - Backward_Selection_Variables: [21, 23, 27]

- **For a = 0.01:**
  - Forward_Selection_Variables: [0, 24, 21, 23]
  - Backward_Selection_Variables: [21, 23, 27]

- **For a = 0.005:**
  - Forward_Selection_Variables: [0, 24, 21, 23]
  - Backward_Selection_Variables: [21, 23, 27]

We notice no change at all between the thresholds, meaning those features have a low p-value and thus are consistently statistically significant.

For the downstream analysis, we chose to use the forward selection features, which are:
0, 24, 21, 23

After we have our selected features, we now have to filter our training and test data based on those features, meaning we only keep the columns indicated in our feature selection results (indices).

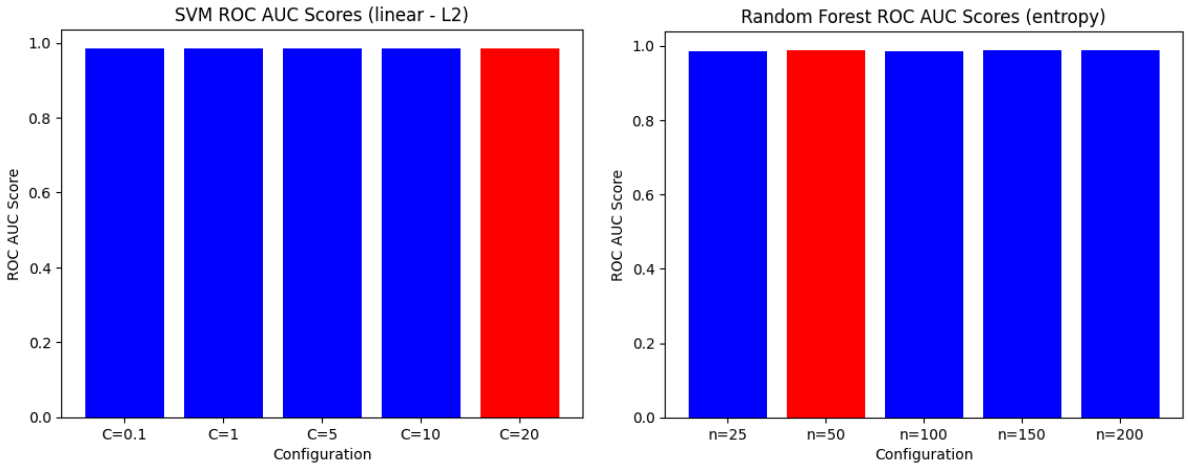
Next step is to set up the parameters for our models.
For Support Vector Machine:
We use SVM with a linear kernel (straight line/ hyperplane as decision boundary). For regularization we use ridge regularization (L2) which penalizes the square of the magnitude of the model coefficients, which prevents overfitting by discouraging overly complex models.
For our hyperparameter C which controls the trade-off between achieving a low training error and a low testing error (controls how much you want to avoid misclassifying each training example), we use the values [0.1, 1, 5, 10, 20]. Small C means we choose a larger-margin separating hyperplane, even if that hyperplane will misclassify more points, leading to higher bias but lower variance. High C means it will choose a smaller-margin separating hyperplane if that means it does a better job of getting all the training points classified correctly, leading to lower bias but higher variance.

For RandomForest:
For our parameters here, we can choose either gini or entropy to check the quality of splits, and we choose entropy, which will use information gain as the criterion for making splits in the decision trees. It basically measures how much each feature contributes by maximizing the overall information gain in the tree.
As for the number of trees in the forest we choose the following values:
[20, 50, 100, 150, 200]. Low number of trees usually leads to faster training but does not always capture the complexity of the data resulting in underfitting. This means higher trees will explain our data in a better manner, but increases computational cost and also comes to plateau after a certain amount of trees.

Since the parameters are now set, we run the cross validation model to choose the best performing parameters. The results as seen in ***results.out*** are:

SVM with params {'kernel': 'linear', 'C': 0.1, 'probability': True}: Average ROC AUC: 0.9842105263157894
SVM with params {'kernel': 'linear', 'C': 1, 'probability': True}: Average ROC AUC: 0.9833849329205366
SVM with params {'kernel': 'linear', 'C': 5, 'probability': True}: Average ROC AUC: 0.9837461300309597
SVM with params {'kernel': 'linear', 'C': 10, 'probability': True}: Average ROC AUC: 0.9843137254901961
SVM with params {'kernel': 'linear', 'C': 20, 'probability': True}: Average ROC AUC: 0.9853973168214655
Random Forest with params {'n_estimators': 25, 'criterion': 'entropy'}: Average ROC AUC: 0.9850877192982456
Random Forest with params {'n_estimators': 50, 'criterion': 'entropy'}: Average ROC AUC: 0.9888028895768833
Random Forest with params {'n_estimators': 100, 'criterion': 'entropy'}: Average ROC AUC: 0.9856553147574818
Random Forest with params {'n_estimators': 150, 'criterion': 'entropy'}: Average ROC AUC: 0.9881836945304437
Random Forest with params {'n_estimators': 200, 'criterion': 'entropy'}: Average ROC AUC: 0.9879256965944272

We notice that we have an almost perfect ROC AUC score for all of them, which means that the parameters seem to be irrelevant, since our data is probably very well behaved (which makes sense for a dataset that is provided for this purpose). If no mistakes were made during calculations, we could choose any one, but we can use the higher one as can also be seen in red in the plots below.

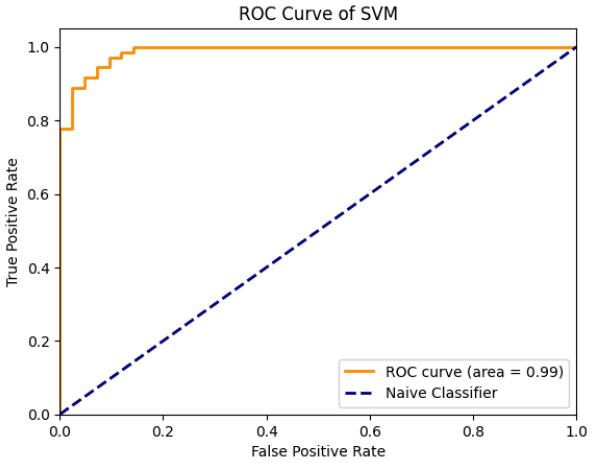


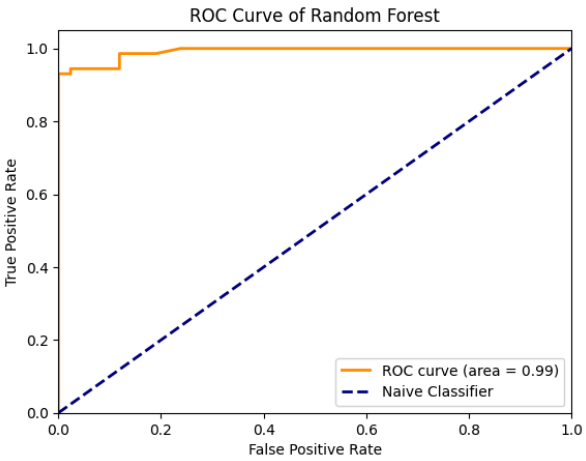
So, we will use SVM with C = 20, and RF with n = 50 (trees in forest).
Using these parameters,we can now train our final model using the train data that has been filtered with feature selection and then evaluate on the filtered held-out set (test set).

As expected, the ROC AUC scores on both models are significantly high, with RF having a slightly higher ROC AUC score on the held-out test set.

- **SVM: ROC AUC on test set:** 0.9877645502645503
- **Random Forest: ROC AUC on test set:** 0.9917328042328042

Let's also calculate the true positive rate (TPR) and false positive rate (FPR) and plot the ROC curve, comparing it with the Naive Classifier.

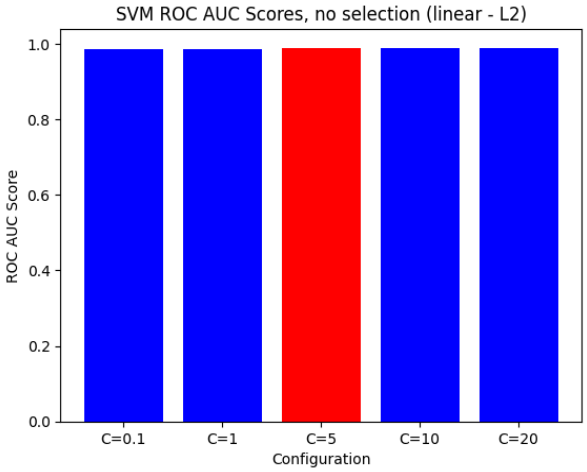


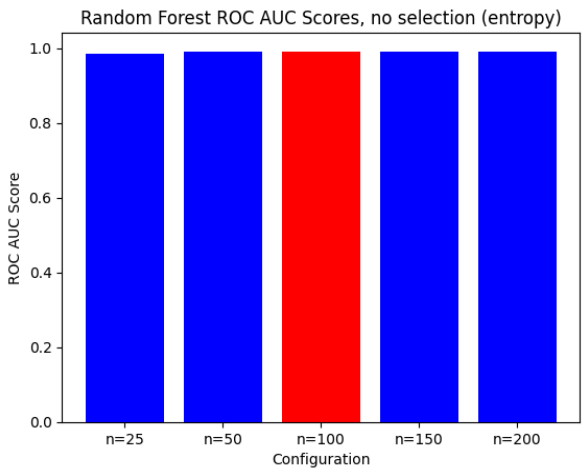


This means our model predicts almost perfectly each sample.
As mentioned before, the selected features for this model were: [0, 24, 21, 23]
We could also grab the names of those features from the original data set:
Feature names: [['mean radius' 'worst smoothness' 'worst texture' 'worst area']

To rerun the classifiers without feature selection, we will perform the same steps, only this time, as training set input, we will use all of the features instead of filtering out with feature selection results. If no mistakes were made during computation again, the results are as following:

SVM with params {'kernel': 'linear', 'C': 0.1, 'probability': True}: Average ROC AUC: 0.986377708978328
SVM with params {'kernel': 'linear', 'C': 1, 'probability': True}: Average ROC AUC: 0.9871001031991744
SVM with params {'kernel': 'linear', 'C': 5, 'probability': True}: Average ROC AUC: 0.9894736842105264
SVM with params {'kernel': 'linear', 'C': 10, 'probability': True}: Average ROC AUC: 0.9893704850361196
SVM with params {'kernel': 'linear', 'C': 20, 'probability': True}: Average ROC AUC: 0.9884416924664603
Random Forest with params {'n_estimators': 25, 'criterion': 'entropy'}: Average ROC AUC: 0.9852941176470589
Random Forest with params {'n_estimators': 50, 'criterion': 'entropy'}: Average ROC AUC: 0.990970072239422
Random Forest with params {'n_estimators': 100, 'criterion': 'entropy'}: Average ROC AUC: 0.991640866873065
Random Forest with params {'n_estimators': 150, 'criterion': 'entropy'}: Average ROC AUC: 0.991124871001032
Random Forest with params {'n_estimators': 200, 'criterion': 'entropy'}: Average ROC AUC: 0.9907636738906088



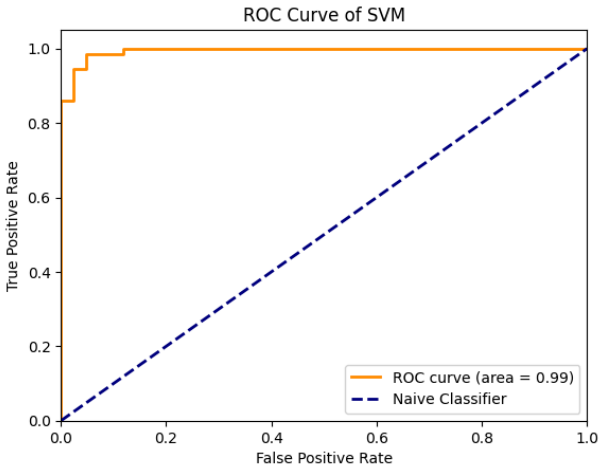


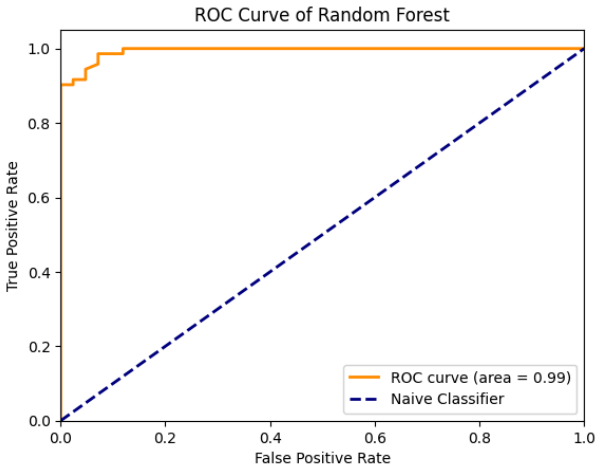
Overall, we notice again almost perfect ROC AUC scores, making almost no difference between the parameters while even including all of the features. A possible explanation for this is that feature selection actually reduced the ROC AUC score, because all of the features (or at least a high number of them) contributed positively to the model's prediction. Also there could be some interaction effect between certain features, that when combined provide a better predictive power. While the values are considerably high, we could also say that feature selection led to overfitting, but again with such high scores that can not be said for certain. We might have a perfectly behaved dataset in this case that by removing features we actually reduce the model's power.

The results for using C = 5 and n=100 as hyperparameters for SVM and RF respectively on the unfiltered held-out set are as following:

- **SVM: ROC AUC on test set:** 0.9943783068783069
- **Random Forest: ROC AUC on test set:** 0.9938822751322751

As expected, we have ever so slightly increased scores than when using feature selection, only this time SVM slightly outperforms RF.





To run forward backward selection on all data, we will use the original dataset D, without splitting into training and test and by using an alpha = 0,05. First we run forward selection, and the results are then used to run backward selection (compared to previously where we used all of the features).
The results are as following (bolded are common features when using training data instead):

- **Forward_Selection_Variables:** [**0**, 6, 15, **21**, 23, 27]
- **Backward_Selection_Variables:** [6, 15, **21**, 23]

We notice more features are selected this time for forward selection, and we have 2 in common. The sets are indeed different, with backward selection excluding features **0** and **27.**
Since backwards selection starts with the set obtained from the forward selection, it will try to exclude any features that will not significantly harm the model's performance. A reason could be that features might have interactions not captured when adding them one by one in the forward selection but are noticed and removed by backwards selection. So for example, if feature 0 or 27 does not contribute additional information or predictive power in the presence of the other selected features, then they might be deemed redundant and removed. This might suggest that while these features can provide some predictive power (alone or in a smaller feature set), they might not offer additional benefits given the combination of features selected by the backward selection process.