ITI104 - Assignment

Introduction

In ensemble learning, predictions from multiple models are combined to improve the performance of the model. These techniques can be applied to both classification as well as regression problems. There are different types of ensemble learning methods and the most commonly used ones are:

- Bagging – this works by sampling on a subset of the data. Separate models are then trained on these samples & the final output is a combination from these different models using averaging or majority voting.
- Boosting – Unlike bagging where training is done in parallel, in boosting training is done sequentially. It starts off with a base classifier, which trains on the entire training data. The next classifier in the pipeline focuses on instances of the training data that the first classifier incorrectly classified. Classifiers are added until the desired accuracy is attained or maximum number of models is reached.

In this assignment, we explore the other class of ensembles - voting and stacking methods. We first describe what voting ensembles are, the different approaches within voting to decide on the final predicted class, when it is appropriate to use them and under what circumstances voting will be effective and their limitations. We then go on to describe the stacking approach, how the algorithm works, their advantages and advantages. Finally, we discuss the pros and cons ensemble methods have over a single classifier/regressor.

In addition, we also discuss the practical assignment in this report, where we demonstrate how we apply the voting and stacking ensemble methods to a suitable toy dataset provided by the Scikit-learn package. The problem chosen is a binary classification problem, using the "Breast cancer Wisconsin (diagnostic) dataset". The idea here is to predict given a set of features, whether the given test data is benign or malignant. I also compare the performance between voting and stacking ensembles as well as the performance of the individual voting and stacking ensemble as well as their base estimators. Here, I use as base estimators the following four algorithms: kNN, SVM, Random Forest and Naïve-Bayes.

Voting Ensembles

Voting ensemble works by combining the predictions from multiple models. It can be used for classification or regression. For example, we can train a dataset using Logistic Regression, support vector machines and naïve-bayes. The output from these three models can then be combined to form an ensemble prediction. The output can be determined by majority voting, i.e., the class that earns the most votes is the preferred class. In the case of a tie, the voting classifier/regressor will select the class based on ascending order. In this way, the resulting class will usually have a higher accuracy compared to using a single algorithm. This is the **hard voting** approach.

So, for example, if we have 5 models and wish to classify whether a test sample belongs to class A or B. Using the 'hard voting' approach, if model 1 predicts that the given test data belongs to class A, model 2 says it belongs to class B, model 3 predicts the test data as belonging to class A, model 4 says class B and model 5 says test data is a class A. By majority vote, the classifier will return the test data as belonging to class A. Figure 1 illustrates how hard voting works.

Hard voting is usually used for models that predict class labels.

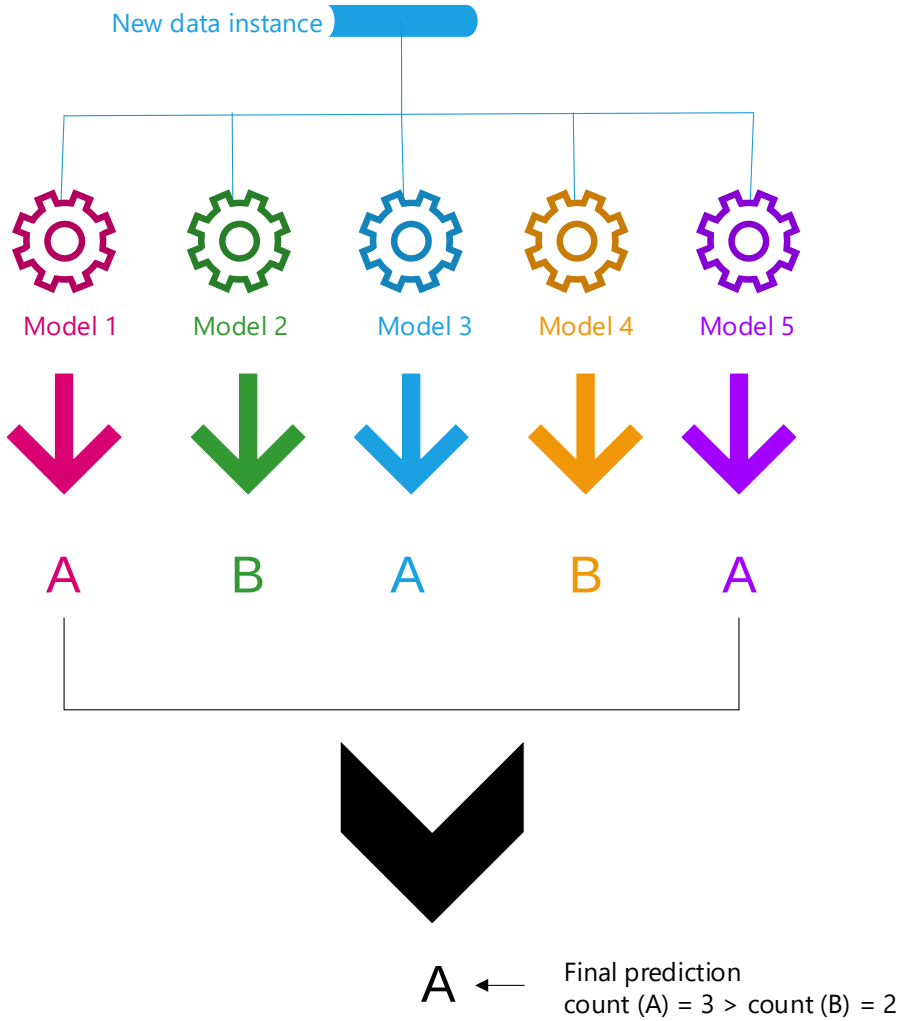


Figure 1 – Voting ensemble – Hard Voting

In **soft voting**, we take an **average of predictions** from all the models and use it to make the final prediction. This is also called averaging. In the averaging or soft voting approach, equal weights are assigned to all the models when computing the output. For **weighted averaging**, we can assign different weights to different models based on their importance. So, if we deem one model more important than another, we assign it a bigger weight. Figure 2 is an example showing soft voting.

Soft voting is useful for models that predict class membership probabilities.

Voting ensembles are appropriate when:

- All models in the ensemble generally have the same good performance.
- All models in the ensemble mostly already agree.

A voting ensemble may not always produce better performance compared to any single model used in the ensemble. If any given model used in the ensemble performs better than

the voting ensemble, then that model should probably be used instead of the voting ensemble.

Voting ensembles are effective under the following circumstances:

- For machine learning models that use a stochastic learning algorithm and result in a different final model each time it is trained on the same dataset.
- When combining multiple fits of the same machine learning algorithm with slightly different hyperparameters.

A limitation of voting ensembles is that it treats all models equally, i.e. all models contribute equally to the prediction. This becomes a problem if a model is appropriate in some situations but does poorly in others.

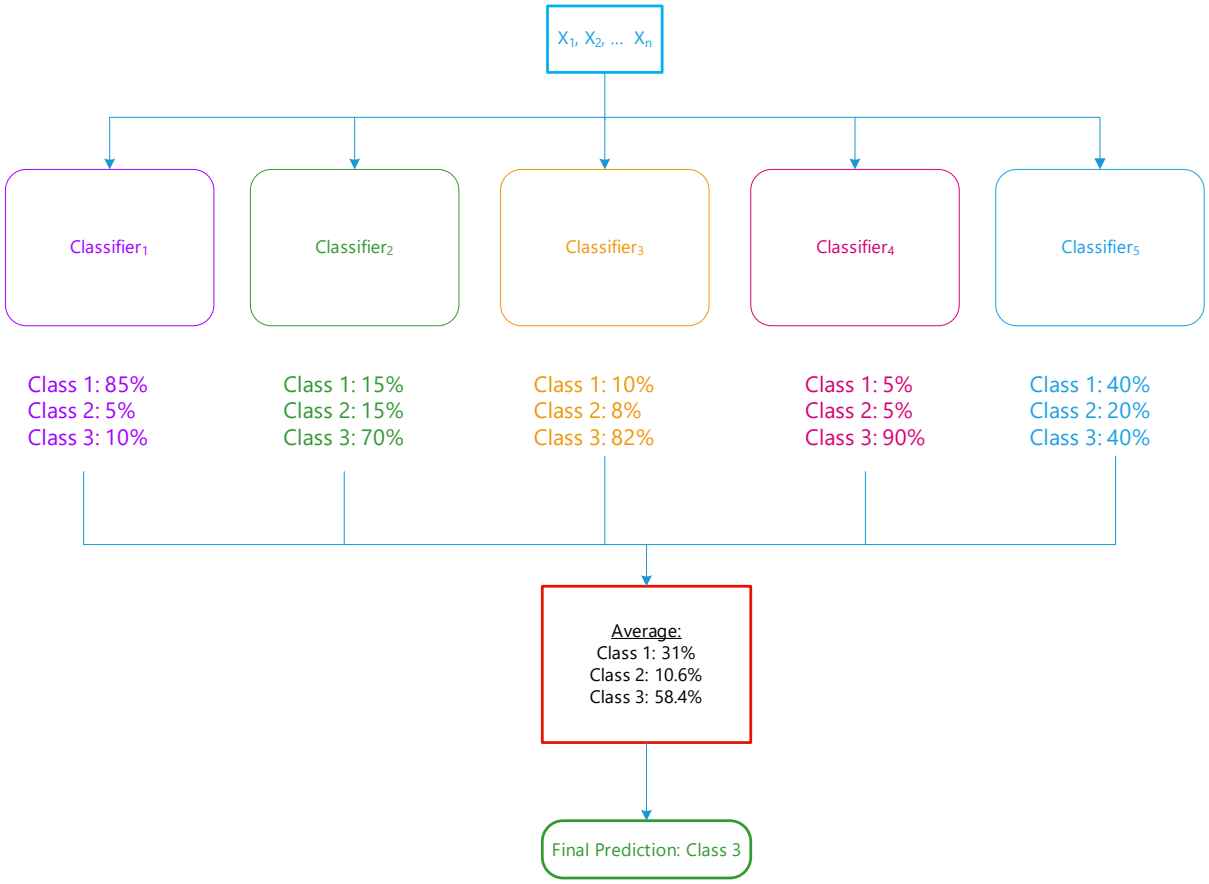


Figure 2 – Voting Ensemble – Soft Voting

Stacking Ensembles

In Stacking, predictions are combined across different machine learning models applied on the same dataset. The first level machine learning models are called the base estimators. The results from these base estimators will be used as input to the final estimator or combiner or meta-learner algorithm. A logistic regression or gradient boosted trees will usually be used as the final estimator. Figure 3 shows an example of a stacking ensemble. Stacking can be used for both classification as well as regression problems.

The stacking algorithm works as follows:

1. The dataset is split into k-folds

2. One fold is chosen as the validation data and the remaining k-1 folds will be used for training.
3. The base models are trained on the training set and predictions are generated for the validation set.
4. Steps 2–3 are repeated for the remaining k-1 folds and an augmented dataset is created with the predictions of each base model included as additional features.
5. The augmented dataset is then trained on the final estimator

Stacking has the following advantages:

- It can yield improvements in model performance.
- Variance is reduced and a more robust model can be created by combining the predictions from multiple models.

It also has the following disadvantages:

- It can take significantly longer to train compared to simpler models and requires more memory.
- Predictions generated using stacked models are usually slower and computationally more expensive. This is important if we plan to deploy such models into production.
- It assumes that the data is independent and identically distributed
- Stacking lacks interpretability because of the modelling ‘gap’ between the inputs and outputs

The difference between voting and stacking is in how the final aggregation is being done. For stacking, a combiner / blender is being done in the final aggregation, whereas in voting average weighting, averaging or majority voting is used to combine the classifiers.

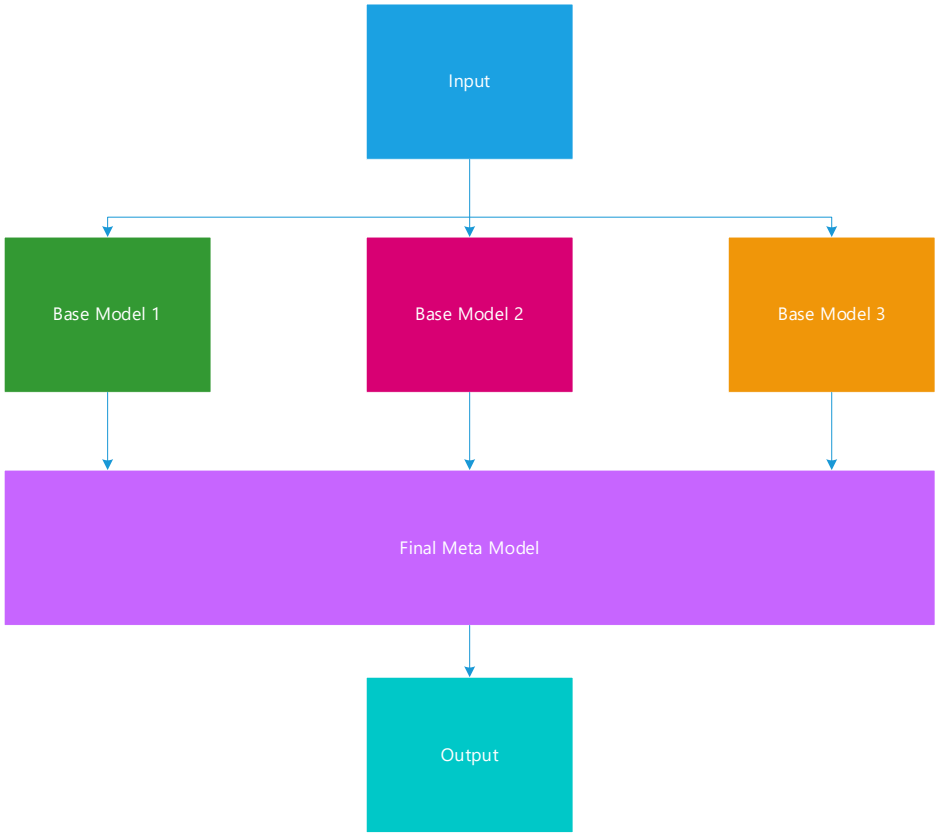


Figure 3 – Stacking Ensemble

Advantages and disadvantages of Ensemble Methods over a single classifier/regressor.

Since ensemble methods comprises a combination of models as opposed to using a single classifier, it usually produces more accurate results. This is evident from recent winners in a number of machine learning competitions such as Kaggle and Netflix. A union of the different models produce stronger results by combining the strengths (and compensating for the weaknesses) of multiple sub-models.

Bias and variance are reduced by incorporating different estimators with different patterns of error, diminishing the impact of a single source of error.

It can also create a deeper understanding of the data.

When it is desired to improve the performance of machine learning models for example, increasing the accuracy of classification models or reducing the mean absolute error of regression models, ensemble methods should be used. It can also result in more stable models.

For single classifiers that overfit on the training set, ensemble methods can be used to create more complex models to overcome the overfitting problem.

They work best when the base models are un-correlated. Each of these models have different strengths and weaknesses. When combined, this may result in a better estimator.

Ensemble models cost more to create, train and deploy. They are also more difficult to interpret. Under certain circumstances, they may not always produce better results, especially if the wrong ensemble method is used e.g., using bagging with a biased model.

Practical Assignment Discussion

For the practical part of this assignment, I have chosen the “Breast cancer Wisconsin (diagnostic) dataset”. This data set has **569** instances and **30** numeric attributes plus the class label.

Attributes
mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error

Attributes
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension

Table 1 – Feature Breakdown of the Breast Cancer (Wisconsin) Diagnostic Dataset

For each image, the mean, standard error, and “worst” or largest (mean of the three worst/largest values) of these features were computed. The features were computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe the characteristics of the cell nuclei present in the image.

Exploratory Data Analysis (EDA)

There are no missing data attributes in this dataset. The data types associated with the features are all 64-bit floating-point numbers. The dataset is fairly evenly distributed, with 357 benign cases and 212 malignant cases. This is shown in Figure 4. The dataset was split into an 80:20 ratio and I also set the stratify parameter to the class label (ground truth). This means that the “train_test_split” method returns training and test subsets that have the same proportions of class labels as the input dataset

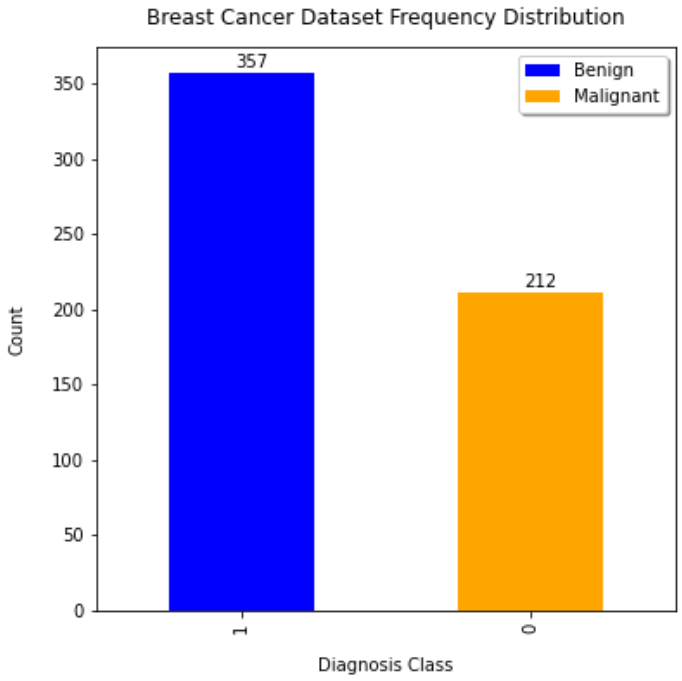


Figure 4 – Breast Cancer Dataset Frequency Distribution

Base Estimators

For all the base estimators, GridSearch was applied in order to determine the optimal parameters to use. Standard scaling was applied so that it has distribution with 0 mean and having a variance equals to 1.

K Nearest Neighbour

For this algorithm, N-neighbours between **1 to 24 inclusive** were tried. The best n neighbour value computed by grid search was found to be **8**.

Hyperparameters	Description	Values
n_neighbours	Number of neighbours to use	1 .. 24

Table 2 – kNN best hyperparameter found

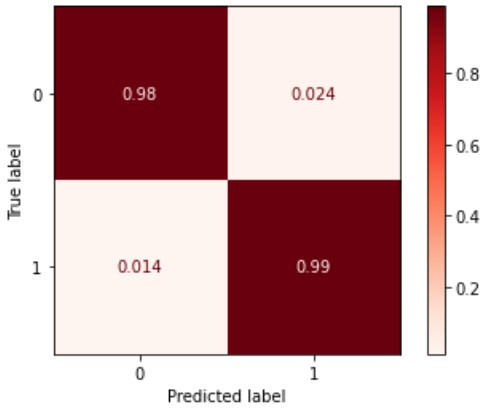
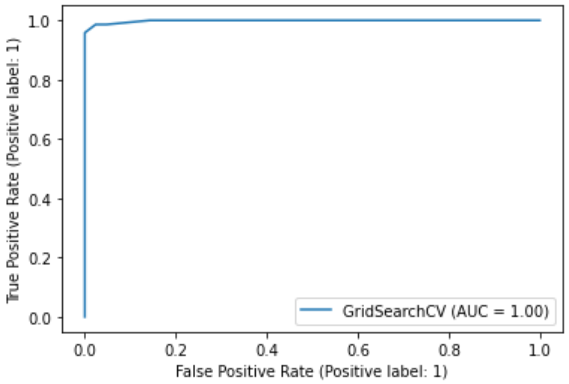
Confusion Matrix	ROC Curve
	

Figure 5 -kNN computed metrics

```
AUC ROC score: 0.9983465608465609
Average Precision score: 0.9985458214624882
Recall score: 0.9861111111111112
```

kNN Classification Report

	precision	recall	f1-score	support
0	0.98	0.98	0.98	42
1	0.99	0.99	0.99	72
accuracy			0.98	114
macro avg	0.98	0.98	0.98	114
weighted avg	0.98	0.98	0.98	114

SVM

For this algorithm, grid search found the following optimal parameters:

C = 10, gamma = 0.01

Hyperparameters	Description	Values
C	C is the penalty parameter, which represents the misclassification or error term. The term tells the SVM optimisation how much error is bearable. This is how you can control the trade-off between the decision boundary and misclassification term. When C is high it will classify all the data points correctly, but there is also a chance to overfit.	0.1, 0.4, 0.6, 0.8, 1, 10, 100
gamma	This controls the distance of influence of a single training point. Low values of gamma indicate a large similarity radius which results in more points being grouped together. For high values of gamma, the points need to be very close to each other in order to be considered in the same group (or class)	1, 0.1, 0.01, 0.001

Table 3 – SVM Hyperparameters

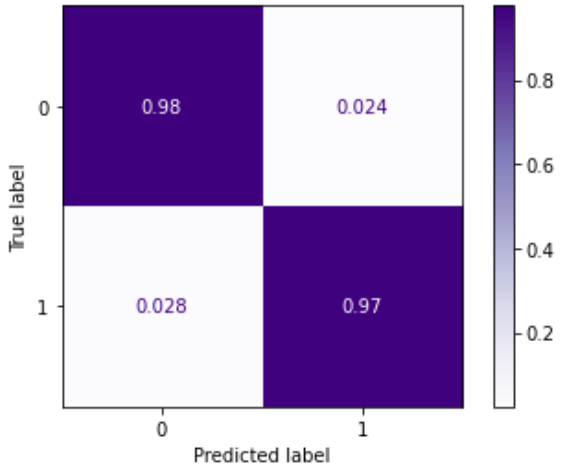
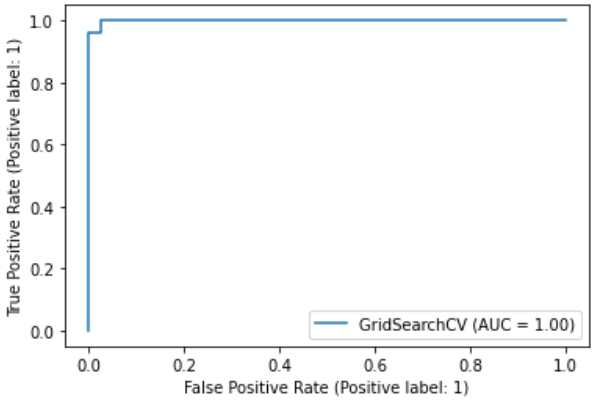
Confusion Matrix	ROC Curve
	

Figure 6 – SVM Computed Metrics

```
AUC ROC score: 0.9990079365079365
Average Precision score: 0.9994212218601648
Recall score: 0.9722222222222222
```

SVM Classification Report

	precision	recall	f1-score	support
0	0.95	0.98	0.96	42
1	0.99	0.97	0.98	72
accuracy			0.97	114
macro avg	0.97	0.97	0.97	114
weighted avg	0.97	0.97	0.97	114

Random Forest Classifier

The optimal number of estimators produced as a result of running Grid search was **1800**.

Hyperparameters	Description	Values
n_estimators	This is the number of trees you want to build before taking the maximum voting or averages of predictions. Higher number of trees give you better performance but makes your code slower.	50, 100, 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000

Table 4 – Random Forest Classifier Hyperparameters

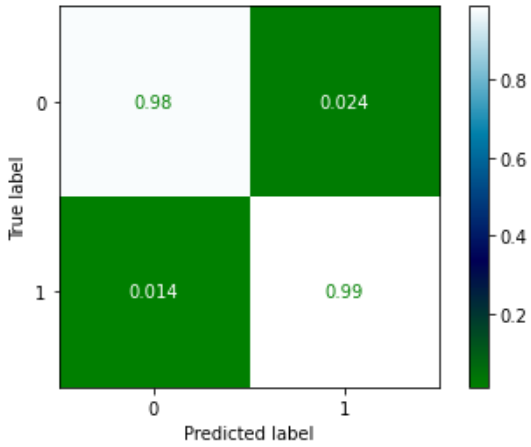
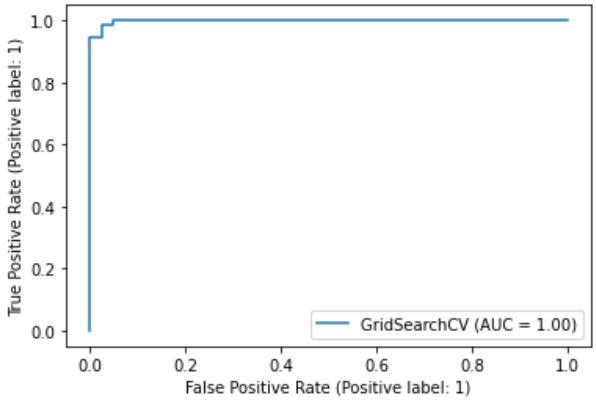
Confusion Matrix	ROC Curve
	

Figure 7 – Random Forest Classifier Computed Metrics

```
AUC ROC score: 0.9983465608465608
Recall score: 0.9861111111111112
Average Precision score: 0.9990376925382796
```

Random Forest Classifier Classification Report

	precision	recall	f1-score	support
0	0.98	0.98	0.98	42
1	0.99	0.99	0.99	72
accuracy			0.98	114
macro avg	0.98	0.98	0.98	114
weighted avg	0.98	0.98	0.98	114

Naive-Bayes

The optimal value for **var_smoothing** was found to be **0.12328467394420659**

Hyperparameters	Description	Values
var_smoothing	This artificially adds a user-defined value to the distribution's variance (whose default value is derived from the training data set). This essentially widens (or "smooths") the curve and accounts for more samples that are further away from the distribution mean.	np.logspace(0,-9, num=100) This returns numbers spaced evenly on a log scale, starts from 0, ends

Hyperparameters	Description	Values
		at -9, and generates 100 samples

Table 5 – Naïve-Bayes Hyperparameters

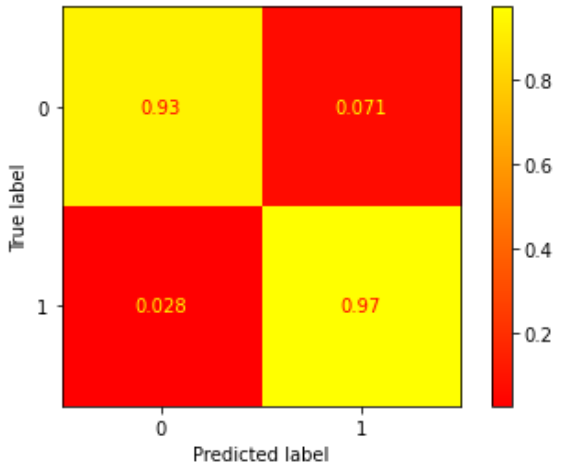
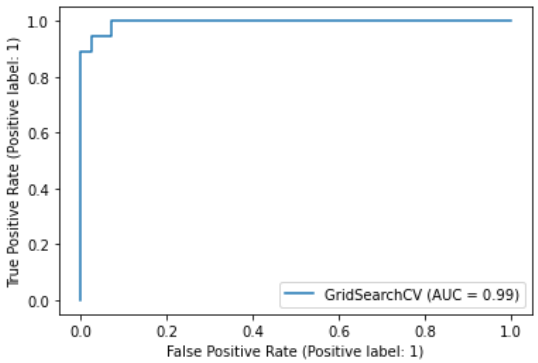
Confusion Matrix	ROC Curve
	

Figure 8 – Naïve-Bayes Computed Metrics

```
AUC ROC score: 0.9947089947089948
Recall score: 0.9722222222222222
Average Precision score: 0.9969086302516648
```

Naïve Bayes Classification Report

	precision	recall	f1-score	support
0	0.95	0.93	0.94	42
1	0.96	0.97	0.97	72
accuracy			0.96	114
macro avg	0.96	0.95	0.95	114
weighted avg	0.96	0.96	0.96	114

Voting Ensemble (soft)

Using the voting classifier and applying ‘soft’ voting, we get the following results:

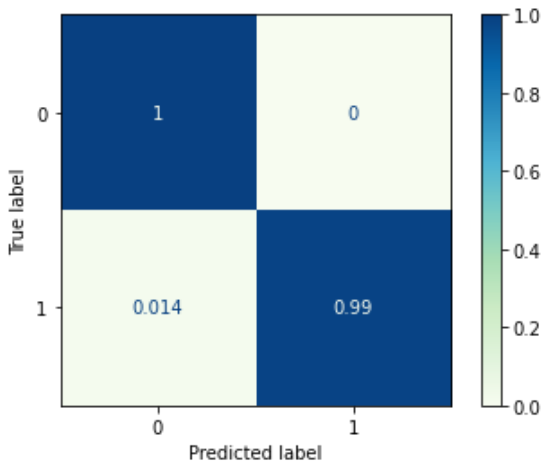
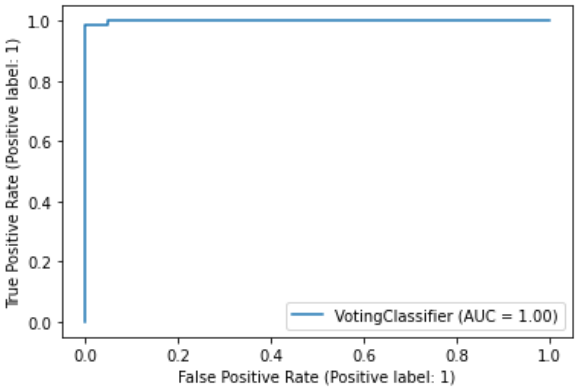
Confusion Matrix	ROC Curve
	

Figure 9 – Voting Ensemble (Soft) Computed Metrics

```
AUC ROC score: 0.9993386243386243
Recall score: 0.9861111111111112
Average Precision score: 0.9996246246246245
```

Voting Classifier (Soft) Classification Report

	precision	recall	f1-score	support
0	0.98	1.00	0.99	42
1	1.00	0.99	0.99	72
accuracy			0.99	114
macro avg	0.99	0.99	0.99	114
weighted avg	0.99	0.99	0.99	114

Voting Ensemble (hard)

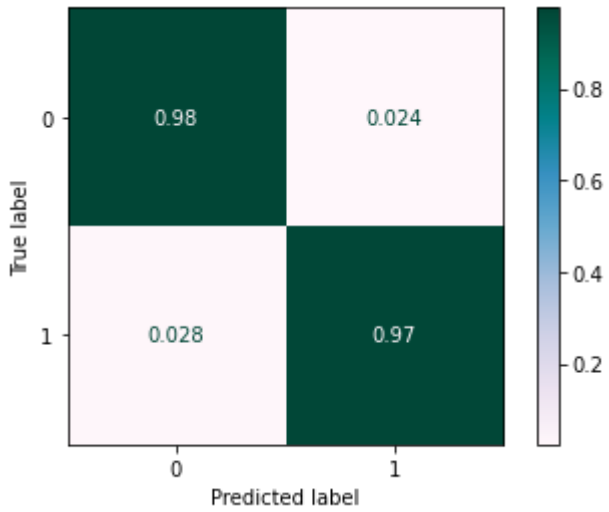
Confusion Matrix	ROC Curve
	N.A

Figure 10 - Voting Ensemble (Hard) Computed Metrics

```
Recall score: 0.9722222222222222
```

Voting Classifier (Hard) Classification Report

	precision	recall	f1-score	support
0	0.98	0.98	0.98	42
1	0.99	0.99	0.99	72
accuracy			0.98	114
macro avg	0.98	0.98	0.98	114
weighted avg	0.98	0.98	0.98	114

Voting Ensemble Summary Scores

I put all the scores into a data frame and the results are shown below:

	score
knn	0.982456
svm	0.973684
rfc	0.982456
nb	0.956140
voting (soft)	0.991228
voting (hard)	0.982456

Figure 11 – Summary Scores for base estimators vs final voting classifier

Applying the **max()** function on the score column, 'voting (soft)' classifier came up tops with a score of **0.9912280701754386**

Stacking Classifier

We use the same set of base estimators like what we have for the Voting ensemble. However, for the stacking classifier, we will use logistic regression as the final estimator.

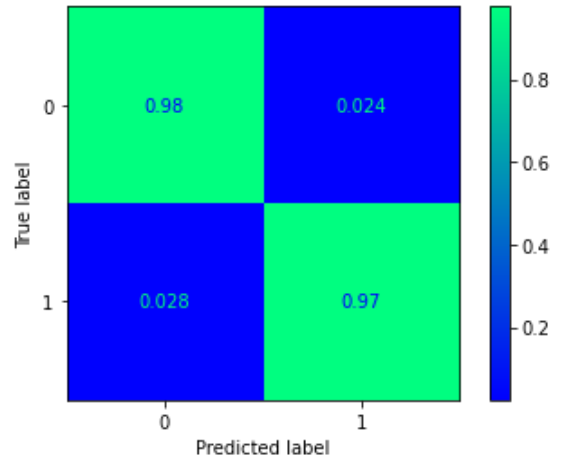
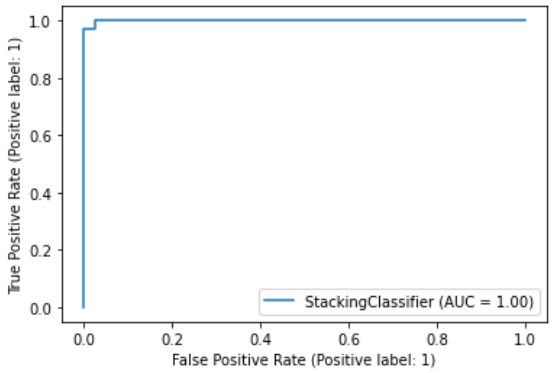
Confusion Matrix	ROC Curve
	

Figure 12 – Stacking Classifier Computed Metrics

```
AUC ROC score: 0.9993386243386243
Recall score: 0.9722222222222222
Average Precision score: 0.9996168400135295
```

Stacking Classifier Classification Report

	precision	recall	f1-score	support
0	0.95	0.98	0.96	42
1	0.99	0.97	0.98	72
accuracy			0.97	114
macro avg	0.97	0.97	0.97	114
weighted avg	0.97	0.97	0.97	114

Stacking Ensemble Summary Scores

I append the stacking score to the data frame created previously.

	score
knn	0.982456
svm	0.973684
rfc	0.982456
nb	0.956140
voting (soft)	0.991228
voting (hard)	0.982456
stacking	0.973684

Figure 13 – Summary Scores for base estimators vs final voting classifier

Applying the **max()** function on the score column, 'voting (soft)' classifier came up tops with a score of **0.9912280701754386.** The voting (hard) ensemble together with k Nearest

Neighbor & the Random Forest classifier comes in a close second (they are a tie), with a score of **0.982456**. The stacking ensemble together with SVM came in third. Naïve-Bayes was the worst performer of the lot.

Estimator	AUC ROC Score	Average Precision Score	Recall Score
kNN	0.998	0.996	0.986
SVM	0.999	1.0	0.972
Random Forest Classifier	0.998	1.0	0.986
Naïve-Bayes	0.995	0.997	0.972
Voting (Soft)	0.999	1.0	0.986
Voting (Hard)	-	-	0.986
Stacking	0.999	1.0	0.972

Table 6 – AUC-ROC, Average Precision and Recall Scores for the different base estimators, Voting and Stacking Ensembles

Since this dataset is fairly balanced, AUC/ROC and average precision are suitable metrics that can be used to evaluate the performance of the models. The high average precision scores in Table 6 tells us that most of the models is able to correctly identify all the positive samples without accidently labelling too many negative samples as positive. AUC/ROC is a good indicator of a model's ability to discriminate between positive vs. negative classes. When AUC = 1, the classifier is able to perfectly distinguish between all the Positive and the Negative classes correctly. If, however, the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives. The AUC/ROC scores in Table 6 shows values close to 1 which indicates that any of these classifiers are already very good at distinguishing between the positive and negative classes.

Conclusion

In this paper, we discussed voting and stacking ensembles, how each of them work, under what circumstances they can be used, their pros and cons. We have also briefly gone over the advantages and disadvantages ensemble methods have over a single regressor or classifier. Finally, we present the analysis of the practical assignment and discuss the results of applying voting and stacking ensembles on the chosen toy dataset from scikit-learn. We show that the soft voting ensemble performs best with kNN, random forest and the voting (hard) ensemble tying at second place.

Reference

[1]. Kaggle Winner's Blog

[2]. Winning the Netflix Prize: A Summary