Answer to graded assignment 2 in DTE-2501 (AI Methods and Applications) about ensemble methods by Abdullah Karagøz

Table of Contents

Introduction

In this assignment I test ensemble methods in classification. Specifically, ensembles of Gaussian Naïve Bayes classifiers.

The goal is to test and show how a simple Gaussian Naïve Bayes classifier performs when used stand-alone versus when used in an ensemble.

I first built all the classes and functions. Then, a test framework was used to compare the performance of the ensemble for different numbers of classifiers, for five different datasets including the Iris dataset. I report the performance results using a single Gaussian NB classifier and ensembles of Gaussian NB classifiers.

I built the classifiers and ensemble myself, but I compare results of my classes with those from Scikit-learn. I haven't used Scikit-learn to implement my classes. Scikit-learn classes are used only for comparison.

Since teachers didn't allow using Numpy, I didn't use Numpy in my code. The only place I used Numpy is when preparing train and test set to Scikit-learn libraries, as it needs Numpy as argument. The API of the classes I built looks a bit similar to Scikit-learn, but they're far from same. My intention wasn't to build classes with same API as classes from Scikit-learn.

I have built 3 classes and 2 functions.

The classes are:

- **Gaussian Naïve Bayes classifier**: The class only accepts data with numerical attributes. The classes must be turned into integers starting from 0.
- **Ensemble Naïve Bayes classifier**: The class instantiates ensembles of my Gaussian Naïve Bayes classifiers. It uses two methods: majority voting and probability aggregation.
- **Metrics class**: A class with static functions that calculates Cross Entropy Loss, accuracy and F1 score.

The functions are:

- **Testing function**: Testing all the classifiers, both single and ensemble methods, using both majority voting and probability aggregation. It tests Gaussian Naïve Bayes classifier and ensemble classifier from Scikit-learn too just for companionate. It prints and compares the results using metrics like mean Cross Entropy Loss, F1 score and accuracy.
- **Plotting function**: This is testing only the ensemble methods, majority voting, probability aggregation, and class from Scikit-learn. It compares from 2 to N number of classifiers how performance changes when you increase number of classifiers in the ensemble method.

In the code I have first built the classes and functions mentioned above. Then I have uploaded datasets and tested them.

1. Classes

1.1 Gaussian Naïve Bayes Classifier

The name of the class is NaiveBayesClassifier. It uses Gaussian probability density function, and it only accepts dataset with numerical attributes.

I have implemented both calculation with probabilities and log probabilities. In my tests they perform the same. A benefit of using logarithmic probabilities is preventing underflow: the calculation of a final or interim numerical value that is so low that the interpreter represents it as zero. Often, when calculating probabilities from a probability distribution function, we obtain exponentially small values. When these are multiplied, they become so small that 64-bit Python's numerical representation is insufficient to capture them and zero values occur. If we use logs to represent these probability values, this can be avoided.

You first initialize the class. Then using fit() function you pass the dataset as the argument. It must be a Pandas Dataframe with numerical attributes only, and classes needs to be integers starting from 0. The classes must be saved on the last column.

The fit() function just calculates the prior-probabilities, mean and standard deviation values and other values that'll be used in the calculations. That's to avoid doing same calculation on every prediction. The standard deviation used is "Bessel corrected" where we divide the value by (N-1), which is common when calculating standard deviation on samples.

1.2 Ensembled Classifier

The name of the class is EnsembledNBClassifier. When initializing it needs a parameter on how many classifiers to ensemble as integer (the name of the parameter is '`nr_of_classifiers`'). It only accepts Gaussian Naive Bayes classifiers I have implemented above. On the fit() function it builds N bootstrapped training data sets (size of N defined when instantiating with parameter '`nr_of_classifiers`') sampled with replacement from the original dataset and of identical size to it. Each classifier is trained from a different bootstrapped dataset.

It has predict and `predict_proba` functions that works as the class above. Both functions use a helper function named ´`__get_all_probs`´ which returns probabilities for each class from each classifier. This list of list of lists is then used in majority voting or probabilities aggregation. This class too has an option to use logarithmic probabilities or not.

There are two bagging methods implemented in this class:

- **Aggregate probability:** For each test data instance, we calculate the probabilities of each class from each classifier. Then for each case for each class, we find the arithmetic mean of all the probabilities calculated from all classifiers. That mean is used as the ensemble probability for the class to be true. Then, we use argmax (choosing the class with highest probability) to convert these probabilities into class predictions. We get the mean-probabilities using `predict_proba` function using a loop like this (inside `predict_proba` function):

```
        for probs in all_probs:
```

```
# Arithmetic mean of all probabilities
agg_probs = [fsum(x)/len(probs) for x in zip(*probs)]
probabilities.append(agg_probs)
```

Then when we have the mean probabilities, getting the predictions using argmax is easy:

```
agg_probs = self.predict_proba(test_set, log_prob)

predictions = [probs.index(max(probs)) for probs in agg_probs]
```

- **Majority voting:** Like a voting system where classifiers are voters and their predictions are their votes, we choose the class that most classifiers has predicted as our predicted value. There may be ties, like two or more classes getting same number of votes. In that case we resolve ties by randomly choosing between these two classes. This is how we count the votes, and use random.random() to resolve ties:

  ```
  pred = max(set(preds), key = lambda x: preds.count(x) +
  0.1*random.random())
  ```

  Since vote count are integer values, adding a value between 0 and 0.1 won't change the vote count, it only make a difference in case of ties. We must resolve ties at random to avoid bias.

Whether to use majority voting or aggregate probabilities is determined when predicting (using 'predict' function) by setting `majority_vote` to True or False.

1.3 Metrics class

A simple class with static functions. The name of the class is Metrics.

While the goal of the classifier is to get as high accuracy as possible, I used other performance metrics too which can give more insight. It has three functions:

- **Cross Entropy Loss:** Accepts probabilities for each class as parameter along with a list of desired values.
- **Accuracy:** Accepts predict and desired results as parameter, just simply calculates accuracy and returns the value.
- **F1 score:** Same parameters as accuracy function, but has a 'macro' Boolean parameter too which is set to 'True' by default. If set True, the function returns mean F1 score. If not then the function returns F1 score for each class. F1 represents the geometric mean of recall and precision and is often a more revealing metric of a classifier's performance than simple accuracy.

There are other metrics to measure performance too, but I only implemented the three above. I think these three gives enough insight on performance of the classifiers.

2. Testing and validation

2.1 Testing function

Function to test and print performance metrics on a datasets.

This function tests Gaussian Naive Bayes and ensemble methods with different methods N times, and compares F1, accuracy and cross entropy loss between them. It prints the results using tabulate table. The tests are:

- Single GNB from Sklearn
- Single GNB classifier without using log probs
- Single GNB with using log probs (I don’t expect with and without log to be different).
- Ensemble from Sklearn
- Ensemble using aggregated probabilities
- Ensemble using majority voting (cross entropy loss not available)

Parameters:

- dataset (Pandas Dataframe): the dataset to train and test on.
- nr_of_classifiers (int): Nr of classifiers ensemble methods will use
- train_test_split (float): The rate of the dataset that will be used for training.
- n (int): Number of tests.
- seed (int): Random state seed.
- log_prob (bool): Whether to use log prob or not on ensemble methods other than Sklearn.

I don’t expect GNB classifier with and without using logarithms to give different results, as using logarithms just gives another scale and avoids underflow. When calculating with log probabilities we use addition and subtraction than multiplication and division. I just included it to confirm whether they give same results or not.

2.2 Function to plot performance of ensembled classes

Function to test and plot results of ensemble methods with different nr of classifiers.

This function tests performance of ensemble methods with Gaussian Naive Bayes classifier, plots the results of the test from 2 to N number of ensembled classifiers. It makes little sense to ensemble one classifier, and it gives different results than a single stand-alone classifier. That is because we use random sample with replacement from the training set, and this makes the training set different from the training set of single classifier alone. This causes some small differences.

The performance metrics used are, F1, accuracy and cross entropy loss.

We compare these 3 classifiers:

- Ensemble from Sklearn

- Ensemble using aggregated probabilities
- Ensemble using majority voting (cross entropy loss not available)

Parameters:

- dataset (Pandas Dataframe): the dataset to train and test on.
- nr_of_classifiers (int): Max nr of classifiers ensemble methods will use.
- train_test_split (float): The rate of the dataset that will be used for training.
- seed (int): Random state seed.
- log_prob (bool): Whether to use log prob or not on ensemble methods other than Sklearn.

It doesn't show Cross Entropy Loss for ensemble method with majority voting. That is because it's not based on probabilities from each classifier, but predictions from each classifier. Also, the predicting each case is based on "votes" from classifiers, where prediction of each classifier is counted as one "vote".

2.3 Testing various datasets

After preparing all classes and functions, now it's easy to test and report performance results.

In all tests the dataset will be split on a 4:1 rate, also 80 % on training and 20 % on testing. The ensemble methods will have **10 base estimators**. Those values can be easily changed by setting parameters.

We use a seed on the random number generator too, that's to make it reproducible.

First, I test on the Iris Dataset as the assignment was about, then I run the test four other datasets.

I have only used small datasets that are clean, so I don't need to preprocess and clean-up it. By clean data I mean it has no NaNs, no missing values etc.

After uploading the data on Pandas Data frame, before running the test I have make small edits so that all classes are represented as integers starting from 0.

2.3.1 Iris Dataset

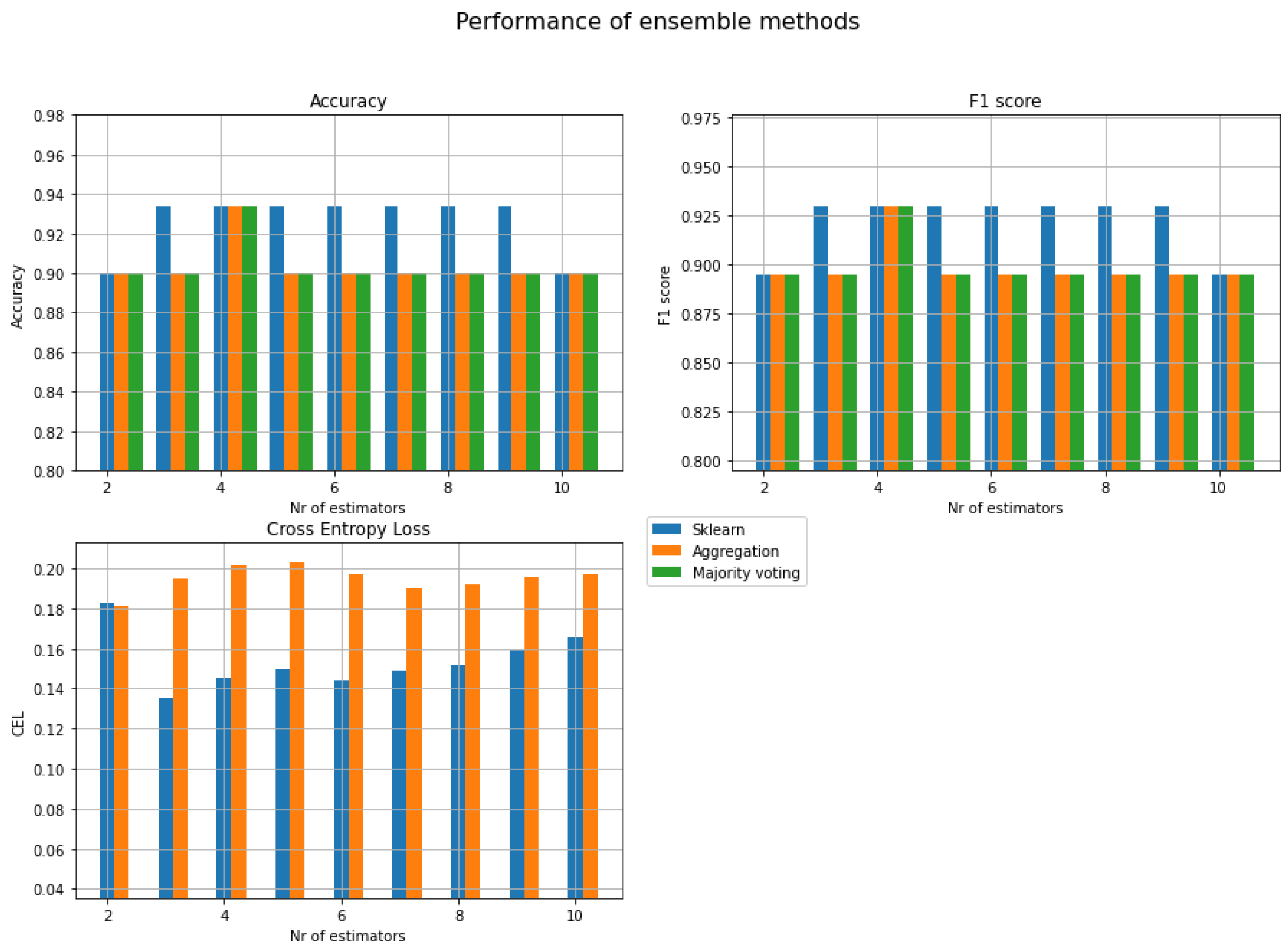
This dataset is about predicting class of Iris plant based on sepal and petal length and width. It has 4 input variables and 3 classes.

Size of dataset: 150
Size of test set: 30
Standard error (1 / (size of test set)): 0.0333 (the error of a single misclassification)

Iris Dataset results

Classifier	**Accuracy**	**F1 Score**	**Cross Entropy Loss**
Single classifier Sklearn	0.9000	0.8947	0.19965
Single classifier	0.9333	0.9296	0.19739
Single classifier with log prob	0.9333	0.9296	0.19739
Ensemble classifier Sklearn	0.9000	0.8947	0.16559
Ensemble classifier with majority voting	0.9000	0.8947	N/A
Ensemble classifier with aggregated probabilities	0.9000	0.8947	0.19688

We see that ensemble methods don't perform better than single classifier in terms of accuracy. But we can also see that the Cross Entropy Loss is reduced with ensemble than with single classifier. But that is only true with Sklearn. The performance of my classifier and ensemble are broadly the same as Scikit-Learn’s.



Here we have plotted the results of accuracy, F1 score and Cross Entropy Loss using Sklearn and my classifier (using probability aggregation and majority voting). The numbers on the X axis show how many base estimators the ensemble methods had. We see that the performance doesn't increase when we increase number of base estimators. (**Note**: the y-axis doesn't start from 0, the differences are smaller than what it looks like visually).

The results also vary depending on the random seed value too. The dataset is very small, the test is even smaller. Thus, different seed values give different results due to the variance of the bootstrap samples.

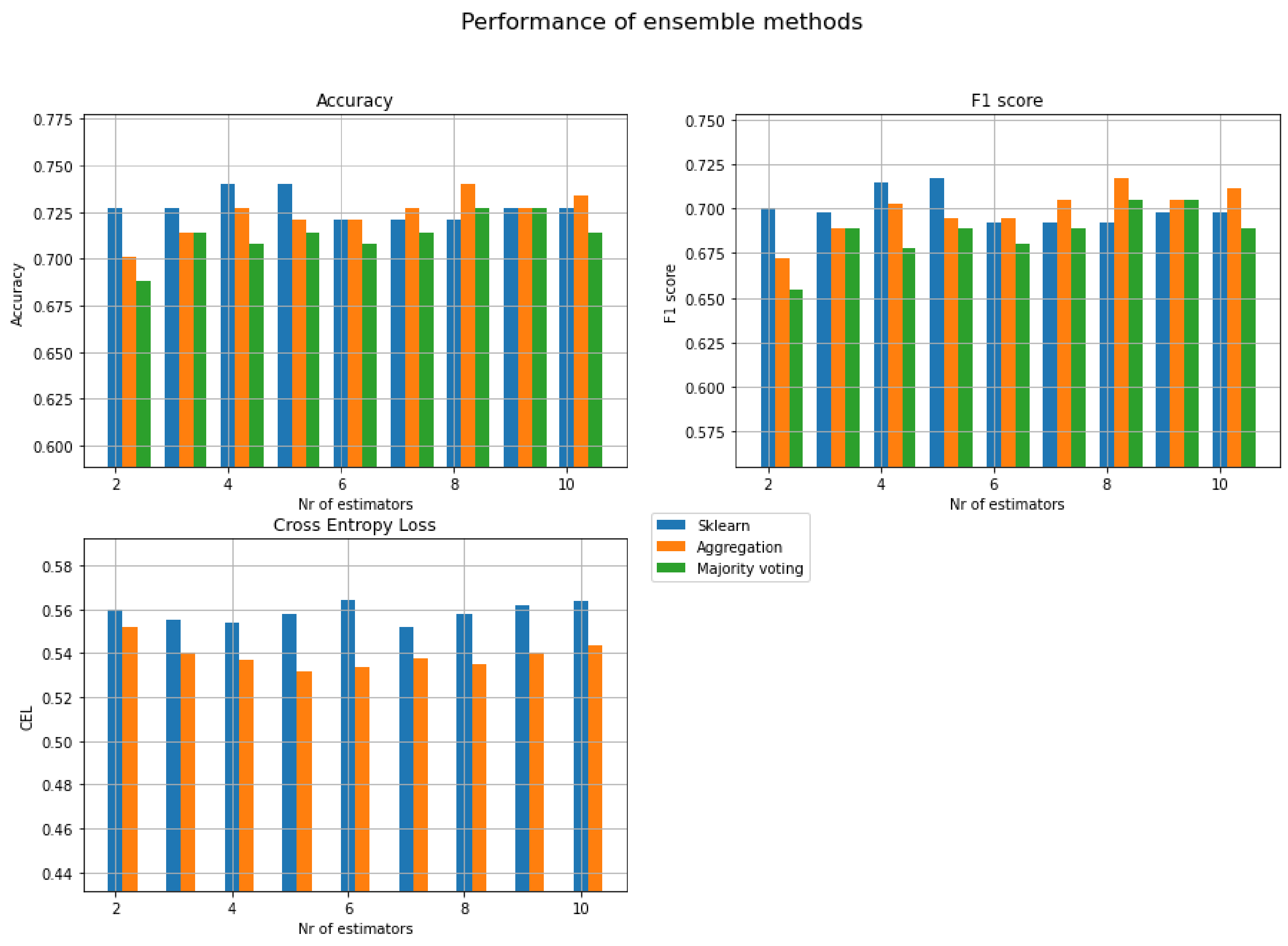
2.3.2 Pima Indians Diabetes Dataset

This is about classifying who has diabetes or not given some medical data. It has 8 input variables and 2 classes (has diabetes or not).

Size of dataset: 768
Size of test set: 154
Standard error (1 / (size of test set)): 0.0065

Classifier	**Accuracy**	**F1 Score**	**Cross Entropy Loss**
Single classifier Sklearn	0.7208	0.6947	0.55633
Single classifier	0.7143	0.6863	0.55698
Single classifier with log prob	0.7143	0.6863	0.55698
Ensemble classifier Sklearn	0.7273	0.6979	0.56340
Ensemble classifier with majority voting	0.7143	0.6889	N/A
Ensemble classifier with aggregated probabilities	0.7338	0.7113	0.54333

Here again we see the results are almost the same. They improve only very marginally by using ensemble classifier. Nor do we observe any improvement in terms of CEL. We see that Sklearn performs slightly better (accuracy and F1) for small ensembles, although its CEL is higher. That may be because Sklearn may use more sophisticated methods in their implementation than my implementation. We see also differences between majority voting and aggregated probabilities too, with the latter performing slightly better.



Here we see that it performs slightly better with increase in the number of estimators. But it's hard to be definitive about that.

2.3.3 Sonar data set

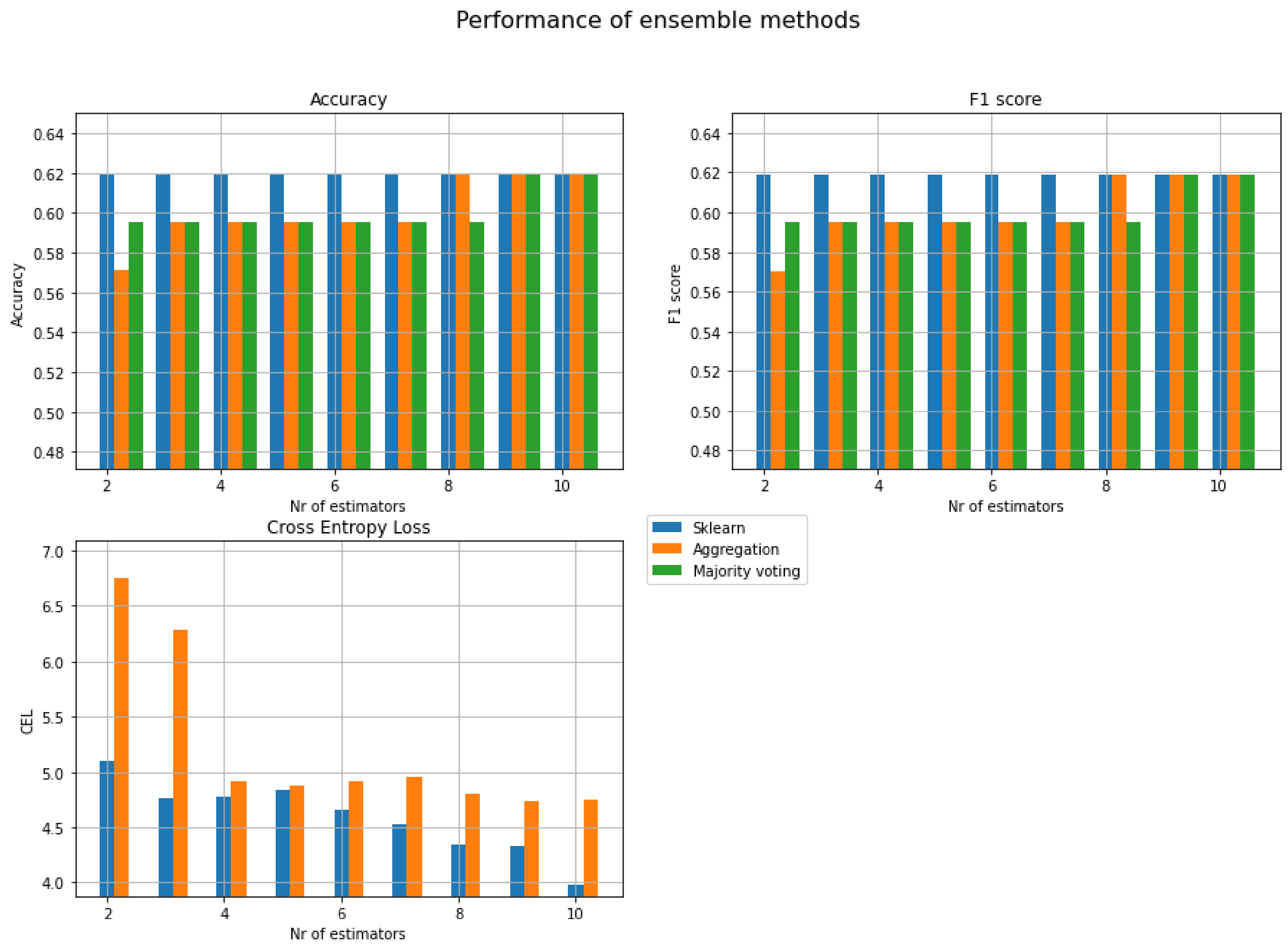
This is about predicting if it's a rock 'R' or a mine 'M' based on some numerical attributes. It has 60 input variables and 2 classes (rock or mine). Those datasets had no column names, and I don't add column names because it's not that important for our analysis.

Size of dataset: 208
Size of test set: 42
Standard error (1 / (size of test set)): 0.0238

Classifier	**Accuracy**	**F1 Score**	**Cross Entropy Loss**
Single classifier Sklearn	0.6190	0.6190	6.55384
Single classifier	0.6190	0.6190	6.47514
Single classifier with log prob	0.6190	0.6190	6.47514
Ensemble classifier Sklearn	0.6190	0.6190	3.97867
Ensemble classifier with majority voting	0.6190	0.6190	N/A

Ensemble classifier with aggregated probabilities	0.6190	0.6190	4.74821

Here we don't see any difference in terms of accuracy or F1, but with ensemble methods it performs better measured by Cross Entropy Loss.



Here we see that increasing number of estimators to ten generally give some better performance, particularly when measured with Cross Entropy Loss. But there is not a linear correlation between number of estimators and performance. Scikit-Learn's classifier performs slightly better (1-2 standard errors) for small ensembles.

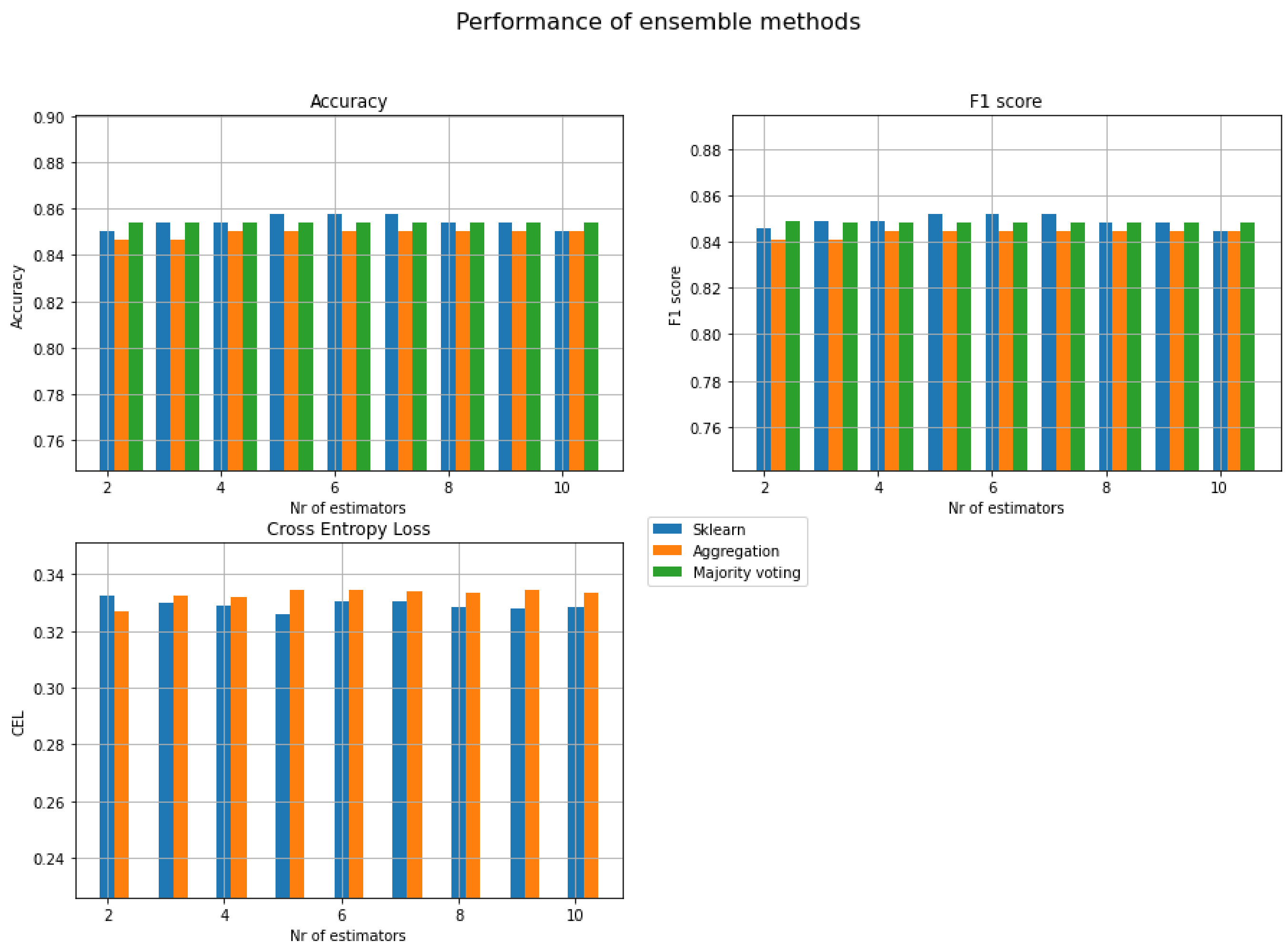
2.3.4 Banknotes dataset

This is about predicting if the banknotes are authentic or not (real or fake) based on some statistics of photographic data. It has 4 input variables and 2 classes (authentic or not).

Size of dataset: 1372
Size of test set: 274
Standard error (1 / (size of test set)): 0.0036

Classifier	**Accuracy**	**F1 Score**	**Cross Entropy Loss**
Single classifier Sklearn	0.8540	0.8481	0.32857
Single classifier	0.8540	0.8481	0.32847
Single classifier with log prob	0.8540	0.8481	0.32847
Ensemble classifier Sklearn	0.8504	0.8445	0.32855
Ensemble classifier with majority voting	0.8540	0.8481	N/A
Ensemble classifier with aggregated probabilities	0.8504	0.8445	0.33356

Here we see the performance differences are very small measured by all three metrics.



The performance difference is small even when tested on ensemble methods with different number of estimators too. It is interesting that this should be the case for the dataset with the largest test set available.

2.3.5 Seeds dataset

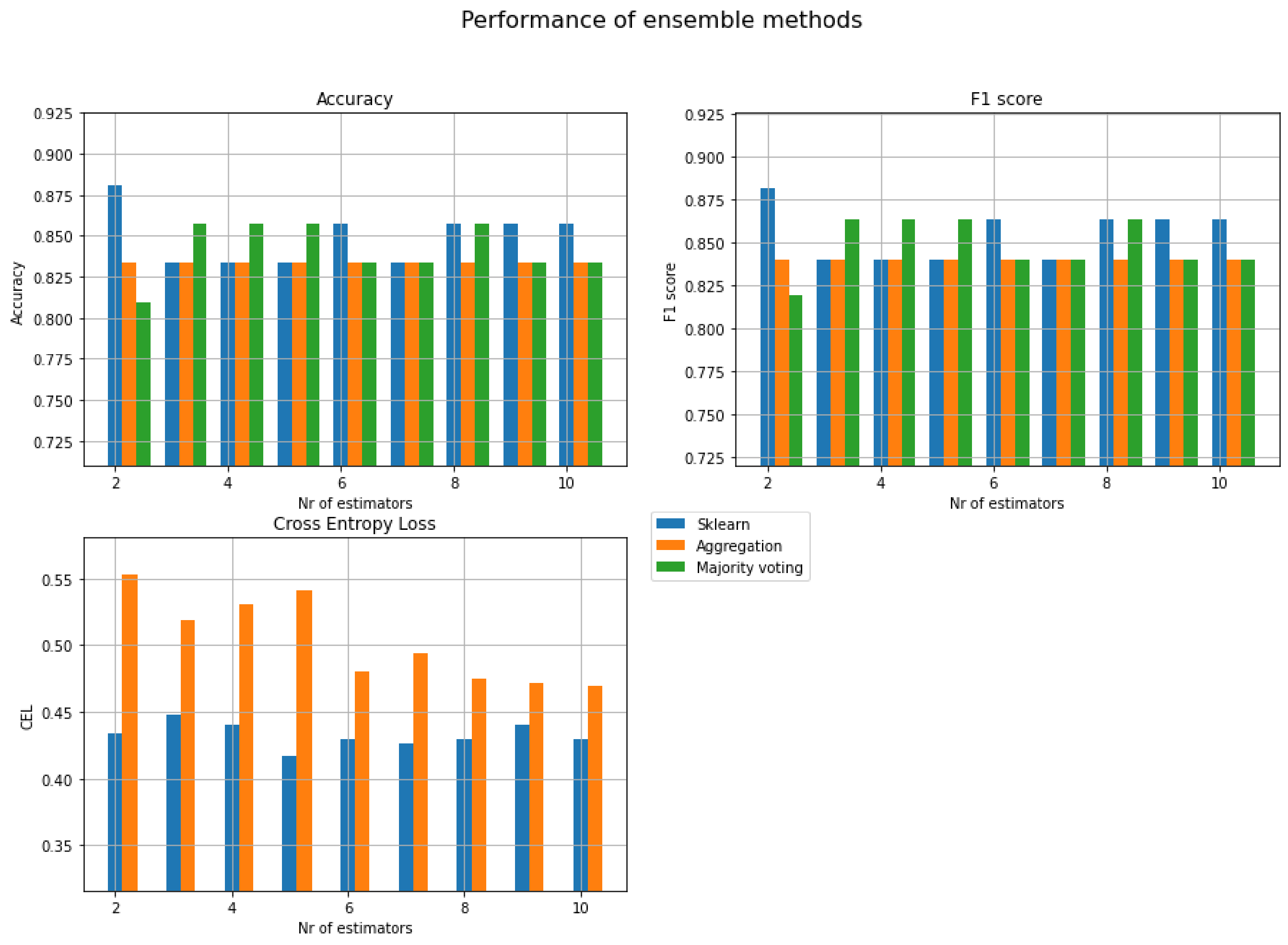
It's about classifying seeds given some attributes. It has 7 input variables and 3 classes.

Size of dataset: 210
Size of test set: 42
Standard error (1 / (size of test set)): 0.0238

Classifier	**Accuracy**	**F1 Score**	**Cross Entropy Loss**
Single classifier Sklearn	0.8571	0.8636	0.53481

Single classifier	0.8571	0.8636	0.52523
Single classifier with log prob	0.8571	0.8636	0.52523
Ensemble classifier Sklearn	0.8571	0.8636	0.42972
Ensemble classifier with majority voting	0.8333	0.8403	N/A
Ensemble classifier with aggregated probabilities	0.8333	0.8403	0.46905

Here again we see almost same performance, and some worse with ensemble classifiers. But measured by Cross Entropy Loss ensemble methods has a slightly higher performance.



We see similar results here too, same or worse results when we increase number of estimators in our classifiers. But in the small datasets the results might be just convenience. That's because we choose random sample with replacement when bootstrapping, we divide the train and test set using random samples too. In small datasets this may make big difference on balances of the test set and training set, and thus give different results.

2.4 Summary of results

As expected, there's no difference between using log probabilities or just probabilities in calculations, or the difference is too small to be significant. Like in case of testing with banknotes when comparing Cross Entropy Loss:

Single classifier	0.32847280381980176
Single classifier with log prob	0.3284728038198017

Just some very minor difference. The cause of this may be something technical on how interpreter calculates than mathematical.

But **generally it's very hard to make general conclusions about our results across the datasets**. Sometimes they perform similarly, sometimes ensembled classifiers perform slightly better or worse. We see differences between my classes and Sklearn's classes too. It may be because Sklearn uses some more sophisticated implementations.

But **usually the differences are very small**. Often only one or two standard errors (the error of a single misclassification). The y-axis on the visualizations don't start from zero, that is to see the differences more clearly.

While I used seed value to make the code reproducible, have run these tests several times without seed too. And the results give slightly variations. There are some parts of the process where random plays role:

- We use random split when splitting dataset into training and testing set.
- In bootstrapping we use random samples with replacement when dividing the training set into several sets.
- We use random choice to resolve ties in majority voting. I suspect it does not play a significant role as it's not very likely, but it still plays some role.

With small datasets the effect of single misclassifications is larger than in bigger datasets. For example, when the test set consists of just 30 samples, a single misclassification can change metric scores more significantly that larger test sets.

Other than random factor, there are other factors that may determine differences in performance:

- How balanced the datasets are, also how skewed the label distributions are. E.g. if there are 2 classes (class A and B), and 90 % of the classes in the set are class A, the dataset is not very balanced.
- Balance of the train set and testing set.
- Regarding difference between the results from Sklearn and my classes despite using same method: Sklearn is professional library and may use more sophisticated implementation methods, while my classes are relatively simple. Even the differences are small it has still effect.
- Number of marginal cases: marginal cases are those in which the probabilities of membership of multiple classes are similar. For example, in a dataset of just two classes, the probabilities of marginal cases might be like 49% / 51%, very close to each other. Those cases are hard to predict. Higher number of marginal cases increase the variance of results because small errors

or perturbations (caused by bootstrap sampling) will cause big changes in classification outcomes. Ensembles could reduce those variances and help perform better. I haven't checked how many marginals the datasets did have.

- Majority voting and aggregated probabilities perform differently sometimes. Majority voting only considers the prediction of each member, the actual probabilities are lost – a prediction of one class might arise from a probability ranging from 51% to 100%. However aggregated probabilities consider the probability (the *confidence*) of each ensemble member. This is, each member's class probabilities are considered in every prediction. In marginal test cases, aggregated probabilities should outperform majority voting because it adds the probabilities and takes their mean value. E.g. if we have a dataset with two classes. One case on class 1 is classified by four classifiers as follows: 1, 1, 0, 0. In majority voting here we have a tie and just use random to resolve it. In probability aggregation we add the probability values. It may be that classifiers that predicted 1 had higher confidence than classifiers predicted 0. Thus, with aggregated probabilities we just predict it as 1. That's just an example of how aggregated probabilities could work better than majority voting.
- But as we see from the tests (on Banknotes dataset) the majority voting gave a little higher accuracy and F1 score than aggregated probabilities. Again, let's say five classifiers predict a case of 1 as 1, 1, 1, 0 ,0. Majority voting would predict it as 1 which is correct. But it's possible the classifiers who predicted it as 0 had higher confidence despite being wrong than classifiers who predicted 1. In that case the aggregated probability would predict it as 0, which is wrong here. That's just an illustration how aggregated probabilities could fail while majority voting predicts it correctly.

3. Conclusions

I expected that ensembled classifiers would always outperform single classifiers in all performance metrics. But this is not the case. At least not when using Gaussian Naïve Bayes classifiers. Sometimes it even performs worse.

I observed that the results vary a lot, because of so much random factors involved in the validation process and ensemble methods, and the datasets I used were very small. With higher datasets this variance is smaller.

Another thing I learned is that GNB is not as bad as I thought. Because it's very simple I thought it should be very bad at predicting values. But it's very efficient when you take into account that the "training" is not some complicated process like in case of neural networks, but just preparing the mean, std and prior probability values.

The reason for why the ensembled Naïve Bayes classifiers doesn't always perform better than single NB classifiers are two things:

- **Naïve Bayes classifier is a high-bias, low-variance classifier.** It means it's common for NB classifier to have high bias error. That is because it uses very simple mathematical formulas to predict classes. This can't represent complex situations or feature interactions. However, the variance of GNB classifiers is comparatively low, because it uses class distribution probabilities, the effect of a single training instance is low. Ensemble methods don't improve when used with such low-variance classifiers because all members of the ensemble act the same way. Therefore, the difference won't be big whether you use single classifier or not. It's similar to that when the whole crowd thinks the same, the crowd doesn't make better predictions than a single person.

- **The dataset lacks marginals.** Marginal values are the cases where the probabilities of a single test instance being a member of two or more classes is very similar. This makes it difficult to predict. E.g. in a dataset of just two classes, the probabilities of marginal cases might be like 49% / 51%, also very close to each other. Those cases are hard to predict. Higher number of marginal cases increase the variance because small errors or perturbations will cause big changes in classification outcomes. Ensembles could reduce those variances and help perform better. It seems like in our datasets there lacks marginal samples. When all cases are easy to predict, the different classifiers don't predict each case differently. Also, a single classifier is as good as ensembled multiple classifiers.

We can compare classifiers classifying dataset to humans solving mathematical (or any) problem set:

If all people think the same way, and use the almost same approach to solve the problems, then it will make no improvement if we add more people to solve the problems, as they will give the same answers anyway.

If the problems are very easy to solve, then adding more people won't make much difference. Here a single person would score 100% and perform as good as multiple people working together.