# Ensemble Approaches in Artificial Intelligence

**Jonathan Helmond**

**University: TU-Sofia**

**Course: Data mining**

**Instructor: Professor A. Efremov**

## ABSTRACT

This work explains advanced algorithms in machine learning. The goal of these is to create models that can handle the bias-variance tradeoff by themselves and can in general produce models a higher accuracy then simple algorithms. Furthermore the possibilities of automated machine learning is introduced.

Keywords: ensemble learning, Bagging, Stacking, Boosting, AutoML

## INTRODUCTION

In nowadays machine learning tasks people are rising for the stars to have the best possible model to complete a task. To reach this goal many model that won competitions are trained with ensemble learning. In this work these approaches are going to be explained. Thanks for using Overleaf to write your article. Your introduction goes here! Some examples of commonly used commands and features are listed below, to help you get started.

## BIAS-VARIANCE TRADEOFF

The focus on machine learning is to find a mathematical model that that represents a uncertain model and can make predictions of it. The uncertain model can also be a mathematical model, but it makes more sense to represent a real or so called tangible model, which has a undetermined outcome. The uncertain model can be represented by a function $f(x)$ whereas the measurement of this model has always a noise. The noise, $\varepsilon$, can be complex. For this work $\varepsilon$ has zero mean and a variance $\sigma$ added to it. So if the uncertain model $f(x)$ is measured the outcome $y$ is

$$y = f(x) + \varepsilon$$

The goal of a representing model $h(x)$ (i.e. of machine learning) is to have the same result as the uncertain model $h(x) = f(x)$. In comparison to the uncertain model, the represented mathematically is defined mathematically. Doe to the noise of a result of a uncertain model model the $h(x) = f(x)$ is impossible. Thus, the goal is to minimize the difference between $f(x)$ and $h(x)$ Belkin et al. (2019). The difference between $f(x)$ and $h(x)$ is called loss $l$ and can be calculated differently. One example of a loss function is the squared loss $l(f(x'),h(x')) = (f(x') - h(x'))^2$.

To create the mathematical model $h(x)$ there has to be a dataset of the uncertain model. This dataset consists is called the training dataset and consists of n input vectors and n outputs $(x_1, f(y(x_1)), ..., ..., (x_n, f(y(x_n))$ which is $\mathbb{R}^d x \mathbb{R}$ where d is the dimension of the input vector. The model $h(x)$ is then build by this dataset. Here is the point where the bias-variance tradeoff comes into play. When choosing the mathematical function for the prediction model there is no perfect function, because the uncertain model can never be predicted to 100%. Also the dataset describing the uncertain model is limited. The challenge is therefore to find the best function for the prediction model. The perfect function has two properties. The first one is bias. A perfect prediction function for bias has almost no bias(i.e. no loss) on the given dataset of the uncertain model. On the other hand this prediction function will have a high bias on new data of the uncertain model and therefore has a high variance. The variance is defined as the size of different loss results when applying the prediction function to new data of the uncertain

model. When a prediction function has a low variance you say that the prediction model is good at generalising the uncertain model.

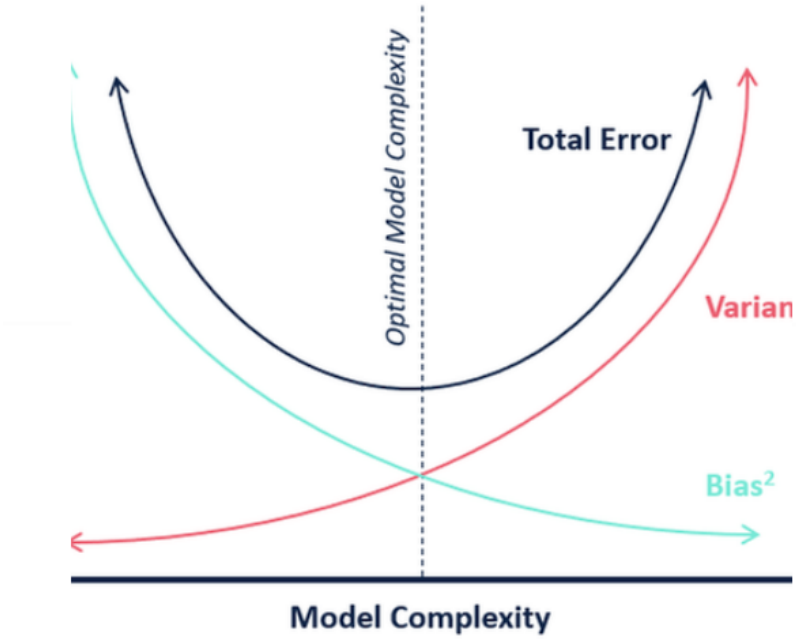



**Figure 1.** The Graph shows the tradeoff between a low bias and a low variance.

To have both, a low bias and a low variance the complexity of the prediction function has to be chosen between adapting to the training dataset and being generalized to make good prediction about unseen data. When a prediction model has a good accuracy(i.e. a small loss) on the training dataset, but a bad accuracy on new data the model is called overfitted. The model adapted too much to the training data and is not able to generalize and validate new data. If a prediction model is performing bad on both, the training dataset and new data it is underfitted. This is illustrated in figure 2.

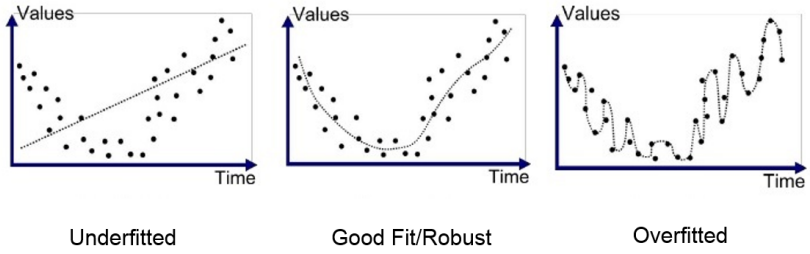



**Figure 2.** The Graph shows the tradeoff between a low bias and a low variance.

### Noise of a real-life model

The noise $\varepsilon$ of a real-life model can be complex. In the moment of measurement there is always the noise added to a uncertain model like $y = f(x) + \varepsilon$. A simple noise would be a normal distribution:

$$\varepsilon(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2}$$

In this case the simplest case is a normal distribution with zero mean and a constant variance. On the other side this has not to be true and the error by noise can be biased which results in a mean unequal to zero. Also the

variance $\sigma$ must not be consistent and can be changing depending on time and input $\sigma(x,t)$, with input $x$ and time $t$.

## ENSEMBLE APPROACHES

One way to of resolving the bias-variance tradeoff is ensemble learning. There are different approaches of ensemble learning for example boosting, bagging, stacking. These approaches are described in the next sections. The idea of ensemble learning is to use multiple different base machine learning models to make a better prediction than a single model would do.

#### *Averaging and Voting*

The way how to make a final result based on several base models depends whether the prediction is a regression or classification task. In regression tasks the final result is made by averaging all the results from the base models. In classification tasks the final result is made by collection the classification results of the base models and deciding for the class with the most votes, whereas each model has one vote for a class.

### Stacking

Stacking is one method to reduce the bias and variance of a prediction model. The idea is to firstly train a pool of base models by different algorithms. A meta model is then trained on the output of the pool of base models.

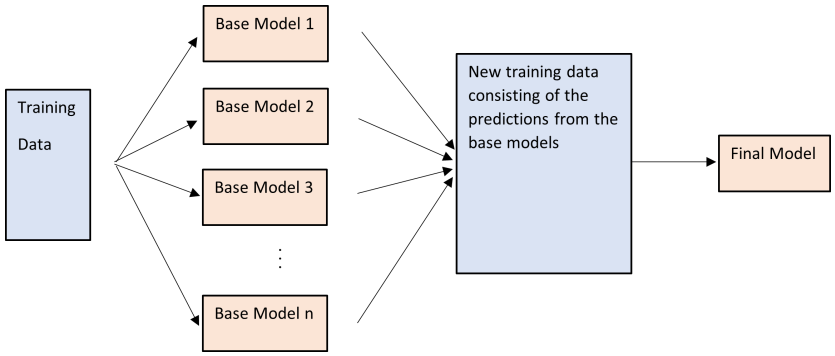



**Figure 3.** This shows the general flow of creating a model with Stacking. The final model is the model performing better than the base models and is therefore the resulting model.

The dataset that the meta model is trained with is commonly created with kfold cross validation to prevent overfitting. As shown in figure 4 the original data is divided into k folds. These folds are used for each model by training each model with cross validation.The Training set is made by cumulating all predictions on the test folds into one new training set. The Validation set/Holdout is generated by making predictions on the validation set/holdout of the original dataset and average the predictions.

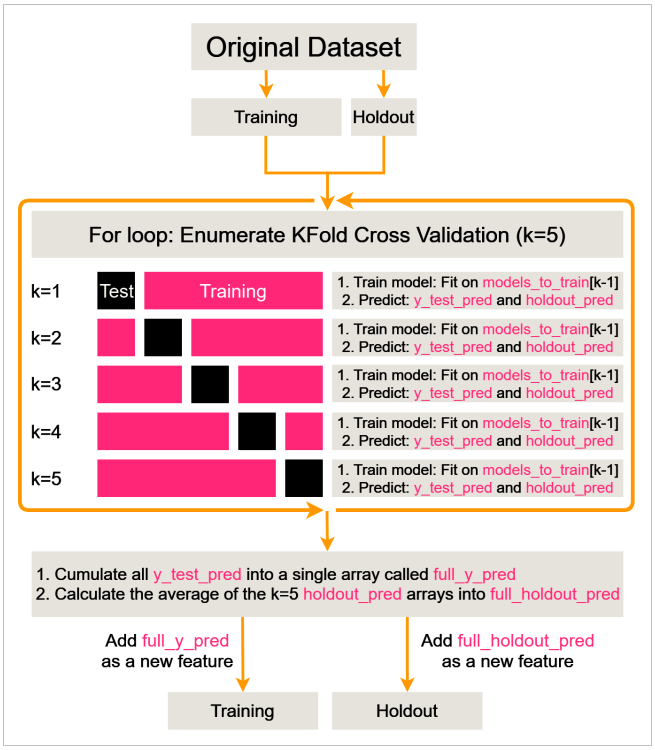



**Figure 4.** This process shows how to create a dataset to train the meta model with. The outcome of this process is a dataset containing a training set and a validation set(i.e. holdout). The image is from Hansen (2020)

The next step is to take the new training and validation set and use them as input for the next level. In a stacking method with two levels this would be the meta model. In a two level stacking method this can be another layer of base learners,. The strategy of multiple base model layers can be different, for example the the first layer can have a great predicting power the second layer can reduce noise. Other purposes of stack layers can be feature selection or normalization. A method of a three layer stacking is illustrated in figure 5

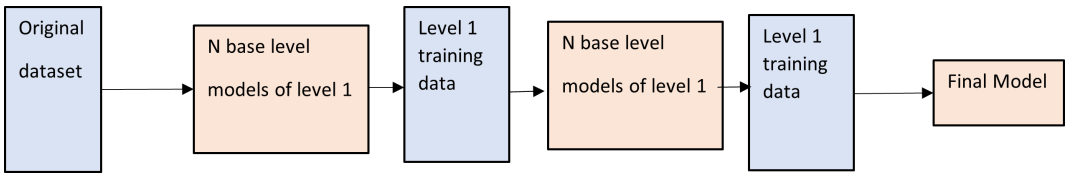



**Figure 5.** In this stacking method there are 2 layers between the original data and the final model.

Stacking works, because it can use the higher dimension decisions of different machine learning algorithms. When used with multiple models, stacked generalization can be seen as a more sophisticated version of cross-validation, exploiting a strategy more sophisticated than cross-validation's crude winner-takes-all for combining the individual models (Wolpert (1992)).

## Bagging

Bagging is the shortage for bootstrap aggregating and consist of these two techniques. It is used to reduce the variance for those algorithm that have high variance, for example decision trees. Before bagging is explained the bootstrap method is explained. Bootstrapping is a statistical method, that belongs to the classes of resampling methods. When applying bootstrapping to a dataset $D$, the new dataset $D'$ will consist of the previous dataset, but some entries are missing and some entries may exist several times. This is done by sampling from D uniformly and with replacement into the new dataset $D'$. The new dataset $D'$ is expected to have the fraction (1 - 1/e) (about 63.2%) of the unique examples of D, the rest being duplicates (Aslam et al. (2007)).

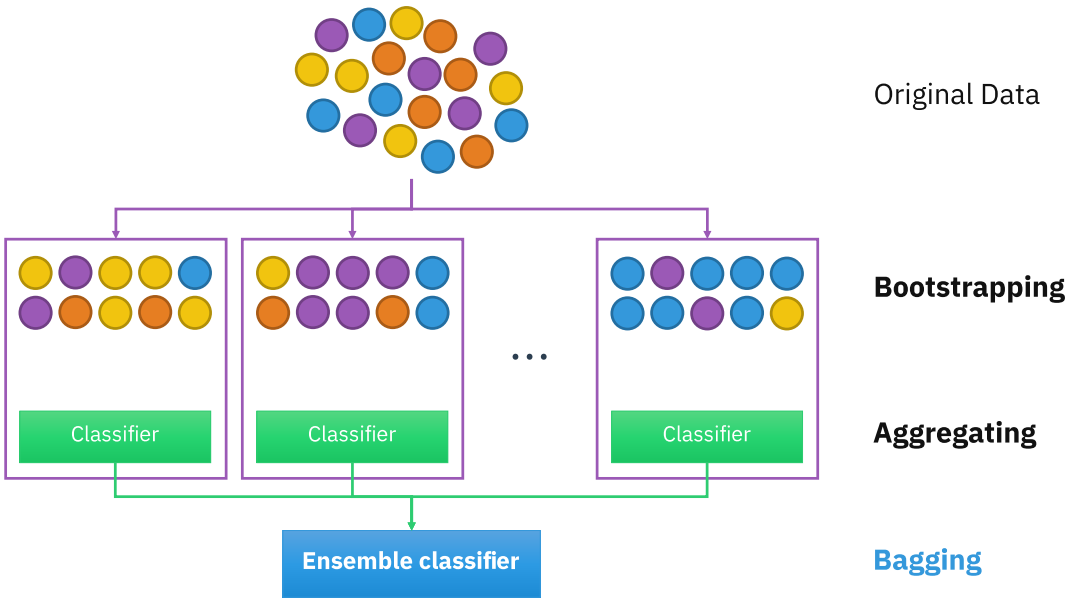



**Figure 6.** The Bagging method uses bootstrapping and aggregating to create one ensemble classifier (image from: Wikipedia contributors (2020a).

The next step of bagging is aggregating. In this step m models are trained each with its own bootstrapped dataset $D'_m$. These m models are then all together the estimator of the purpose question. By averaging the output (for regression) or voting (for classification), like described in chapter , the overall model makes its predictions.

After bootstrapping data, the new distribution of the data has a Gaussian distribution (Aslam et al. (2007)). This has to be kept in mind when using this method, because when using the data for training the model the noise of the data or the distribution of features must be a Gaussian distribution too.

### Boosting

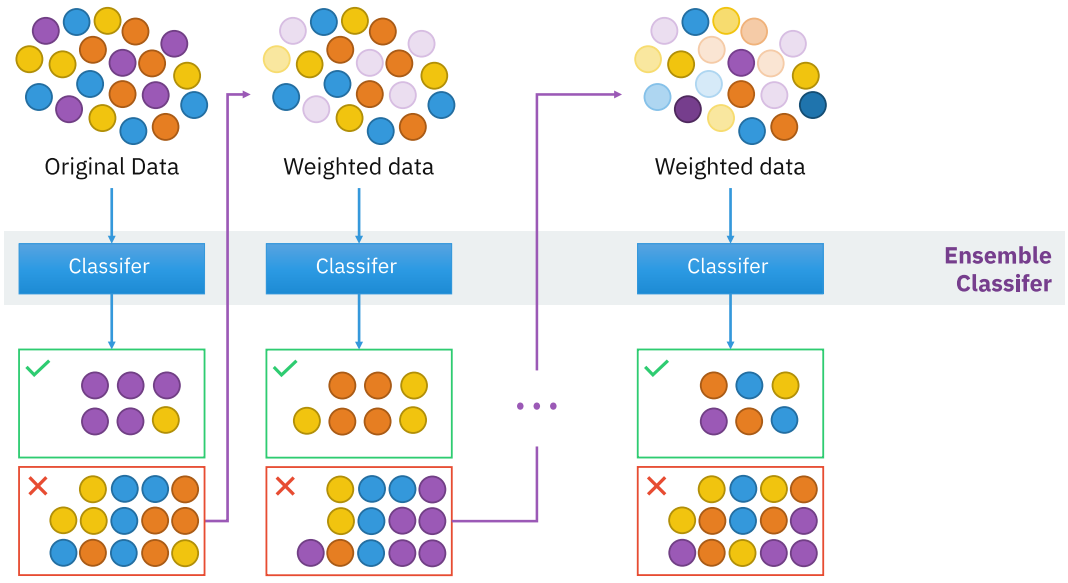



**Figure 7.** (image from: Wikipedia contributors (2020b).

Boosting is a category of optimisation algorithms to optimize a learning algorithm for machine learning tasks. The basic idea behind boosting is to use multiple weak models with a low accuracy to generate one model with a high accuracy. During the training process the result of the previous generated weak model is used to generate

the next weak model, as shown in figure 7.

In the following is described how the algorithm works. As explained in chapter 1 we have a dataset with input features set X and the output label set Y. For now the set $Y = \{-1, +1\}$. During the process a base learning algorithm is called repeatedly in a series of rounds t = 1, ... , T. Each data $x_i, y_i$ of the dataset is coupled to a weight $W_i$. Initially the weight of all data tuples is the same with having m total tuples $(x_i, y_i)$ the weight for each tuple starts with $W_i = 1/m$. After this the algorithm loops the following steps for t = 1,...,T.

1. Train weak model using the distribution
2. Get the hypothesis of the model $h_t : X -> \{-1, +1\}$ with error $\eta_t = P_{i\ D_t}[h_t(x_i) \neq y_i]$
3. Calculate the amount of say $a_t = \frac{1}{2} * ln(\frac{1-\eta_t}{\eta_t})$
4. Update the weights $D_i = D_i * e^{-a_t}$ if $h_t(x_i) = y_i$ and $D_i = D_i * e^{a_t}$ if $h_t(x_i) \neq y_i$
5. Normalize the new weights $D_i$ so that all weights sum up to 1

The final hypothesis is the weighted sum of all weak models and is calculated like the following:

$$H(x) = sign(\sum_{t=1}^{T} a_t * h_t(x))$$

What is happening in the loop, is that the weights are adapted every round, so that data tuples that were classified correctly have a decreasing weighting for the next round and data tuples that are classified wrong have a increasing weight for the next round. Thus, data that is hard to predict is more important for the learning model of the next round. In the end all the prediction of all models are summed with the amount of say that corresponds to the particular model. the amount of say refers to how good the model is to make predictions.

## AUTOMATED MACHINE LEARNING (AUTOML)

In AutoML the goal is to automatize the generation of machine learning models to real world problems. A complete AutoML covers the complete pipeline from the raw dataset to the finished machine learning model. The steps necessary, which are the same as a non automated process, to achieve the whole task of automation are the following:

1. Data preparation - eg. column data type detection
2. feature engineering - eg. feature importance selection
3. model selection
4. hyperparameter optimization
5. evaluate results

It is also possible to only automate a single or some steps in this process. The amortisation begins with data preparation.

### Automated Data Preparation

In data preparation one of the first steps that can be automated is outlier detection and removal. There are several algorithms that can do this.

#### *Outlier Detection*

Unsupervised algorithms for anomalies detection are for example the Isolation forest the OneClass SVM or a outlier detection with the deviation. The detected anomalies can be classified as outliers. To check whether a removing of these outliers is a good idea, there can be trained 2 models, one with and one without the outliers. If the model without the outliers is better at generalizing than the model trained with the outliers, the outliers should be removed for training.

### Data Scaling

Scaling the data to a standart or normalize them can also be automated. The reason for scaling is often to adapt the data to the model it will be used for. Also data can be scaled into a new space, for example to reduce the dimensions. This makes sense if the dataset is not small, because with less dimensions all calculations will be quicker. One way to do this is to use Principal component analysis(PCA). PCA will find new vectors in the space of attributes that explain the recent dataset the best. In this way having a 20 dimensional dataset the first 5 vectors of a PCA can already describe 98% of the dataset. In case all recent attributes have a effect on the data and maximum accuracy is important PCA should been used. Another possibility is to check for the correlation of the attributes with the target attribute, including also the sum of all attributes with the target attribute as a validation score. For attributes having no correlation with the target attribute it is possible to remove them from the dataset. The sum of correlations with the target attribute can be used to figure out which scaling method should be used. These can be linear, normalisation, logarithmic, exponential and more. For the best scaling method the sum of correlations will be the highest.

#### *Automated Model Selection*

Another automation discussed in this work is the model selection and hyperparameter selection.

The selection of a model can be made upon different statistic statement. These can be whether there is a high linear correlation in the data, whether there are periodically trends, time variation or more. Depending on these statistics characterisations of the data some models are predestined to work better on the data then others. Still the actual deep structure is unknown or most of the time impossible to grasp which is the reason to use a machine model in the first place. So even with a good statistical research it is not possible to say which machine learning model will actually work the best in the end. To find the best model the most promising models have to be trained on the data and evaluated about their predictions. During training the hyperparameter of the learning algorithm must already be optimized, because models can have big differences in accuracy depending on their hyperparameters. To do hyperparameter optimization there are different possibilities, from which grid-search and Bayesian optimization are explained in the next chapters.

#### *Grid Search*

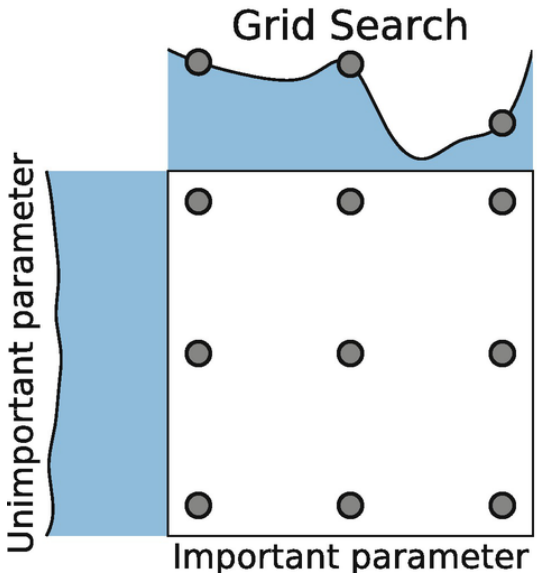



**Figure 8.** Grid search does a sweep over the parameters to hopefully find the optimum with one of the tries. All the points have the same distance to each other. (image from: Lee (2019).

Grid search is a traditional way of finding the best hyperparameters. In grid search a manually set up subset of hyperparameters is sweeped trough to find the best hyperparameters. After the model is trained with each set of the subset the model with the best predictions is used for the future.

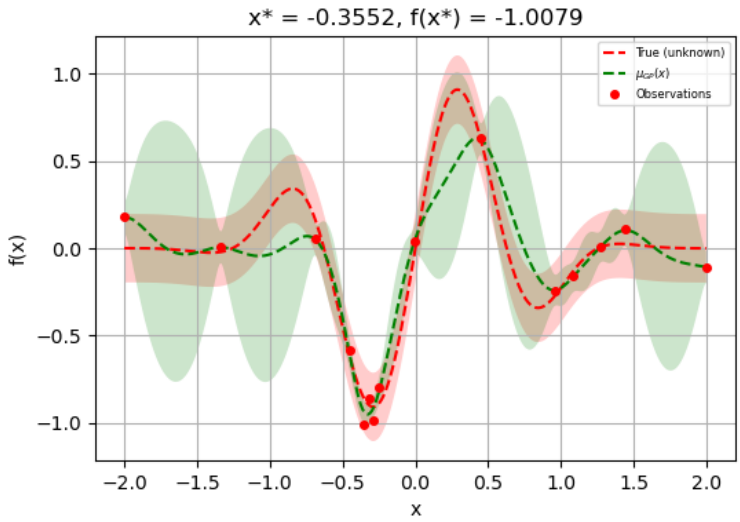



**Figure 9.** The graph shows the true data curve in red, observations that are made with red dots and the prediction of the real unknown curve in green. The green shadow represents the predicted space in which the green line can shift because of to less observations made. (image from: Lee (2019).

#### *Bayesian optimization*

Bayesian optimization is a optimization process that that uses prior knowledge gained from former model training's. So the cycle of one training is longer than for example with grid search where the next input is preknown and not calculated after the last cycle. Bayesian optimization builds a probabilistic model of the function mapping from hyperparameter values to the objective evaluated on a validation set. The space of hyperparameters is step for step evaluated and a optimum is found by predicting the space between the observations and searching for the most promising predicting areas.

#### *Comparison*

Both optimization methods are reasonable depending on the task. For a small task grid search is the better choice because of its simplicity. Also in practice, Bayesian optimization has been shown Hutter et al. (2011) to obtain better results in fewer evaluations compared to grid search and random search, due to the ability to reason about the quality of experiments before they are run. Still Bayesian optimization is effective in finding the absolute optimum and not a point close to it.

## CONCLUSION

In this work the automated machine learning is introduced and different ensemble learning algorithms are explained. automated machine learning is nowadays common and the whole process of building a model is getting more and more automated. It is foreseen that in the future models can be created completely automated ready for deployment. Ensemble learning algorithms are really strong algorithms can can produce models that are robust to outliers, have a high accuracy and are robust to over- and underfitting. On the other side these algorithms need more calculation resources and also more memory to be stored. In times where computers are getting more powerful and memory gets cheaper every year these algorithms should be considered for every good model building.

## ACKNOWLEDGMENTS

I want to thank my covid quarantaine to make me work more than I would have done if I would have been allowed to go out.

## REFERENCES

Aslam, J. A., Popa, R. A., and Rivest, R. L. (2007). On estimating the size and confidence of a statistical audit. *EVT*, 7:8.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854.
Hansen, C. (2020). Stack machine learning models - get better results.
Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer.
Lee, E. (2019). Medium article. [Online; accessed 20-11-2020].
Wikipedia contributors (2020a). Wikipedia, the free encyclopedia. [Online; accessed 26-10-2020].
Wikipedia contributors (2020b). Wikipedia, the free encyclopedia. [Online; accessed 26-10-2020].
Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5(2):241 – 259.

## CODE

# ensemble_methods_notebook_loans

November 20, 2020

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, f1_score, confusion_matrix,␣
 ↪recall_score

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import BaggingClassifier
```

[2]:
```
#read the data into Datframe
df=pd.read_csv("D:/Sofia_courses/DataMining/loan_data.csv", delimiter=",")
```

[3]:
```
#give the column with multiple possible features each a new column to provide␣
 ↪the same distance of uncorrelated/unlinear features
cat_feats=['purpose']
df = pd.get_dummies(df,columns=cat_feats,drop_first=True)
```

[4]:
```
#show information about the values in the columns
df.describe()
```

[4]:

| | credit.policy | int.rate | installment | log.annual.inc | dti |
|---|---|---|---|---|---|
| count | 9578.000000 | 9578.000000 | 9578.000000 | 9578.000000 | 9578.000000 |
| mean | 0.804970 | 0.122640 | 319.089413 | 10.932117 | 12.606679 |
| std | 0.396245 | 0.026847 | 207.071301 | 0.614813 | 6.883970 |
| min | 0.000000 | 0.060000 | 15.670000 | 7.547502 | 0.000000 |
| 25% | 1.000000 | 0.103900 | 163.770000 | 10.558414 | 7.212500 |
| 50% | 1.000000 | 0.122100 | 268.950000 | 10.928884 | 12.665000 |
| 75% | 1.000000 | 0.140700 | 432.762500 | 11.291293 | 17.950000 |
| max | 1.000000 | 0.216400 | 940.140000 | 14.528354 | 29.960000 |

| | fico | days.with.cr.line | revol.bal | revol.util |
|---|---|---|---|---|
| count | 9578.000000 | 9578.000000 | 9.578000e+03 | 9578.000000 |
| mean | 710.846314 | 4560.767197 | 1.691396e+04 | 46.799236 |
| std | 37.970537 | 2496.930377 | 3.375619e+04 | 29.014417 |
| min | 612.000000 | 178.958333 | 0.000000e+00 | 0.000000 |
| 25% | 682.000000 | 2820.000000 | 3.187000e+03 | 22.600000 |
| 50% | 707.000000 | 4139.958333 | 8.596000e+03 | 46.300000 |
| 75% | 737.000000 | 5730.000000 | 1.824950e+04 | 70.900000 |
| max | 827.000000 | 17639.958330 | 1.207359e+06 | 119.000000 |

| | inq.last.6mths | delinq.2yrs | pub.rec | not.fully.paid |
|---|---|---|---|---|
| count | 9578.000000 | 9578.000000 | 9578.000000 | 9578.000000 |
| mean | 1.577469 | 0.163708 | 0.062122 | 0.160054 |
| std | 2.200245 | 0.546215 | 0.262126 | 0.366676 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 50% | 1.000000 | 0.000000 | 0.000000 | 0.000000 |
| 75% | 2.000000 | 0.000000 | 0.000000 | 0.000000 |
| max | 33.000000 | 13.000000 | 5.000000 | 1.000000 |

| | purpose_credit_card | purpose_debt_consolidation | purpose_educational |
|---|---|---|---|
| count | 9578.000000 | 9578.000000 | 9578.000000 |
| mean | 0.131760 | 0.413134 | 0.035811 |
| std | 0.338248 | 0.492422 | 0.185829 |
| min | 0.000000 | 0.000000 | 0.000000 |
| 25% | 0.000000 | 0.000000 | 0.000000 |
| 50% | 0.000000 | 0.000000 | 0.000000 |
| 75% | 0.000000 | 1.000000 | 0.000000 |
| max | 1.000000 | 1.000000 | 1.000000 |

| | purpose_home_improvement | purpose_major_purchase |
|---|---|---|
| count | 9578.000000 | 9578.000000 |
| mean | 0.065671 | 0.045625 |
| std | 0.247720 | 0.208682 |
| min | 0.000000 | 0.000000 |
| 25% | 0.000000 | 0.000000 |
| 50% | 0.000000 | 0.000000 |

```
75%                    0.000000                0.000000
max                    1.000000                1.000000

       purpose_small_business
count             9578.000000
mean                 0.064627
std                  0.245880
min                  0.000000
25%                  0.000000
50%                  0.000000
75%                  0.000000
max                  1.000000
```

### 0.1 Information about the columns

-credit.policy: 1 if the customer meets the credit underwriting criteria of LendingClub.com, and 0 otherwise.
-purpose: The purpose of the loan (takes values "credit_card", "debt_consolidation", "educational", "major_purchase", "small_business", and "all_other").
-int.rate: The interest rate of the loan, as a proportion (a rate of 11% would be stored as 0.11). Borrowers judged by LendingClub.com to be more risky are assigned higher interest rates.
-installment: The monthly installments owed by the borrower if the loan is funded.
-log.annual.inc: The natural log of the self-reported annual income of the borrower.
-dti: The debt-to-income ratio of the borrower (amount of debt divided by annual income).
-fico: The FICO credit score of the borrower.
-days.with.cr.line: The number of days the borrower has had a credit line.
-revol.bal: The borrower's revolving balance (amount unpaid at the end of the credit card billing cycle).
-revol.util: The borrower's revolving line utilization rate (the amount of the credit line used relative to total credit available).
-inq.last.6mths: The borrower's number of inquiries by creditors in the last 6 months.
-delinq.2yrs: The number of times the borrower had been 30+ days past due on a payment in the past 2 years.
-pub.rec: The borrower's number of derogatory public records (bankruptcy filings, tax liens, or judgments).

[5]:
```
#display some values
df.head()
```

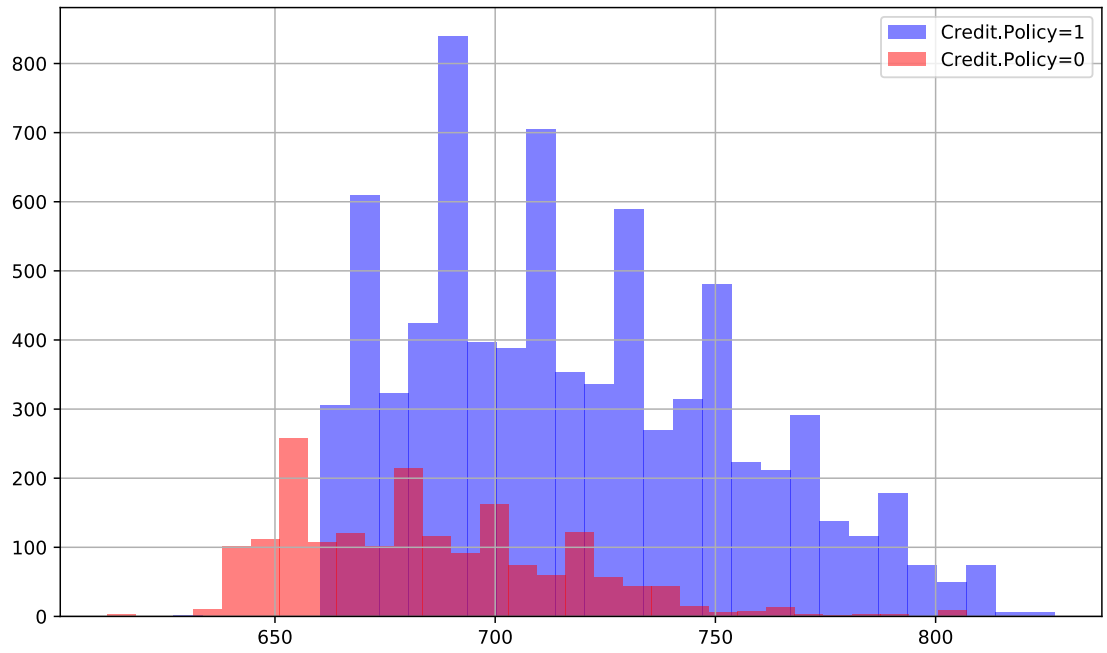
[5]:
```
   credit.policy  int.rate  installment  log.annual.inc    dti  fico  \
0              1    0.1189       829.10       11.350407  19.48   737
1              1    0.1071       228.22       11.082143  14.29   707
2              1    0.1357       366.86       10.373491  11.63   682
3              1    0.1008       162.34       11.350407   8.10   712
4              1    0.1426       102.92       11.299732  14.97   667

   days.with.cr.line  revol.bal  revol.util  inq.last.6mths  delinq.2yrs  \
```

```
0        5639.958333      28854        52.1                0            0
1        2760.000000      33623        76.7                0            0
2        4710.000000       3511        25.6                1            0
3        2699.958333      33667        73.2                1            0
4        4066.000000       4740        39.5                0            1

   pub.rec  not.fully.paid  purpose_credit_card  purpose_debt_consolidation  \
0        0               0                    0                           1
1        0               0                    1                           0
2        0               0                    0                           1
3        0               0                    0                           1
4        0               0                    1                           0

   purpose_educational  purpose_home_improvement  purpose_major_purchase  \
0                    0                         0                       0
1                    0                         0                       0
2                    0                         0                       0
3                    0                         0                       0
4                    0                         0                       0

   purpose_small_business
0                       0
1                       0
2                       0
3                       0
4                       0
```

as visible the rating of the lending club(credit policy) correlates with the value of the FICO

[6]:
```
#show some data

plt.figure(figsize=(10,6))
df[df['credit.policy']==1]['fico'].hist(alpha=0.5,color='blue',
                                        bins=30,label='Credit.Policy=1')
df[df['credit.policy']==0]['fico'].hist(alpha=0.5,color='red',
                                        bins=30,label='Credit.Policy=0')
plt.legend()
```

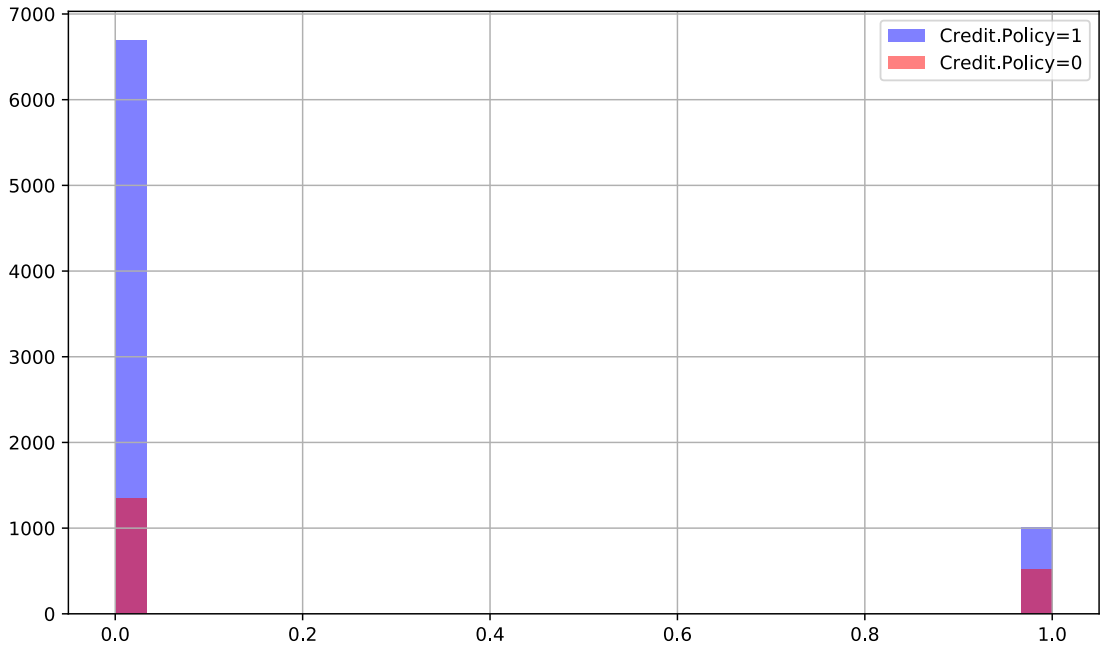
[6]:
```
<matplotlib.legend.Legend at 0x1c6ded5f320>
```




as shown in this chart a costumer fullfilling the critera of the lendingclub (credit.policy=1) have a relative higher chance of paying of their loan

[7]:
```
plt.figure(figsize=(10,6))
df[df['credit.policy']==1]['not.fully.paid'].hist(alpha=0.5,color='blue',
                                                  bins=30,label='Credit.Policy=1')
df[df['credit.policy']==0]['not.fully.paid'].hist(alpha=0.5,color='red',
                                                  bins=30,label='Credit.Policy=0')
plt.legend()
```

[7]:
```
<matplotlib.legend.Legend at 0x1c6df251198>
```




A problem as visible in the last charts is that The credit policy is not evenyl divided into 2 calsses. This can cause problemslater for a model by always predicting the class that is more frequent than the other.

[8]:
```
print("ralative value of the different classes")
(len(df.loc[df["credit.policy"]==0])) / (len(df.loc[df["credit.policy"] == 1]))␣
↪* 100
```

```
ralative value of the different classes
```

[8]:
```
24.228274967574578
```

[9]:
```
#divide the data into training and test set

X = df.drop('credit.policy',axis=1)
Y = df['credit.policy']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30,␣
↪random_state=1)
```

A first check how well a simply linear classifier would work for this task.

[10]:
```
def get_scores(classifier):
    print("accuracy: ",classifier.score(X_test,y_test))
    predicts = classifier.predict(X_test)
    print("F1 score: ",f1_score(y_test, predicts))
    print("")
```

```
    print("Confusion matrix showing false/positives:")
    print(pd.DataFrame(confusion_matrix(y_test, predicts)))

    return f1_score(y_test, predicts)

def just_get_scores(classifier):
    predicts = classifier.predict(X_test)
    return f1_score(y_test, predicts)
```

[11]:
```
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression( max_iter = 100).fit(X_train, y_train)
reg_f1 = get_scores(reg)
```

```
accuracy:  0.9025748086290883
F1 score:  0.9422442244224423

Confusion matrix showing false/positives:
     0     1
0  310   224
1   56  2284
```

[12]:
```
from sklearn.ensemble import RandomForestClassifier
rfc= RandomForestClassifier(n_estimators=400)
rfc.fit(X_train,y_train)
get_scores(rfc)
```

```
accuracy:  0.9940848990953375
F1 score:  0.9963744934954148

Confusion matrix showing false/positives:
     0     1
0  521    13
1    4  2336
```

[12]:
```
0.9963744934954148
```

[13]:
```
from sklearn.tree import DecisionTreeClassifier
dtree =DecisionTreeClassifier(max_depth = 12)
dtree.fit(X_train,y_train)
get_scores(dtree)
```

```
accuracy:  0.9906054279749478
F1 score:  0.9942221271132035

Confusion matrix showing false/positives:
     0     1
0  524    10
1   17  2323
```

[13]: 0.9942221271132035

# 1 Stacking

For stacking some base learners are defined and tested individually for their performance. In the next step they are stagged and the overall performace of the meta learner is checked.

[14]:
```
def get_models():
        models = dict()
        models['logisticRegression'] = LogisticRegression()
        models['knn'] = KNeighborsClassifier()
        models['svm'] = SVC()
        models['bayes'] = GaussianNB()
        return models
```

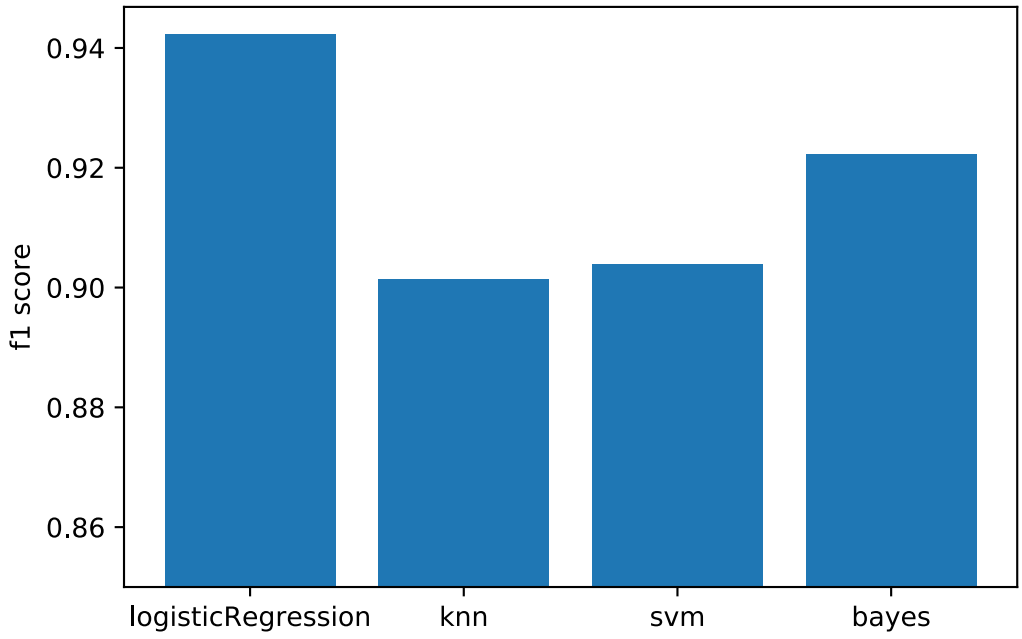
[15]:
```
#get the models
models = get_models()
results, names = list(), list()

def evaluate_model(model):
    model.fit(X_train,y_train)
    return get_scores(model)

#evaluate each model
for name, model in models.items():

        score = evaluate_model(model)
        results.append(score)
        names.append(name)
        print("score of ", name, "is  > ", score)
```

```
accuracy:  0.9025748086290883
F1 score:  0.9422442244224423

Confusion matrix showing false/positives:
     0     1
0  310   224
1   56  2284
score of  logisticRegression is  >  0.9422442244224423
accuracy:  0.8291579679888657
F1 score:  0.9013065326633166

Confusion matrix showing false/positives:
     0     1
0  141   393
1   98  2242
score of  knn is  >  0.9013065326633166
```

```
accuracy:  0.8274182324286709
F1 score:  0.9038013964313422

Confusion matrix showing false/positives:
    0     1
0  48   486
1  10  2330
score of  svm is  >  0.9038013964313422
accuracy:  0.8660403618649966
F1 score:  0.922175055589246

Confusion matrix showing false/positives:
     0     1
0  208   326
1   59  2281
score of  bayes is  >  0.922175055589246
```

[16]:
```
samller_results = [x-0.85 for x in results]
plt.ylabel("f1 score")
plt.bar(names,samller_results,bottom = 0.85)
```

[16]: <BarContainer object of 4 artists>



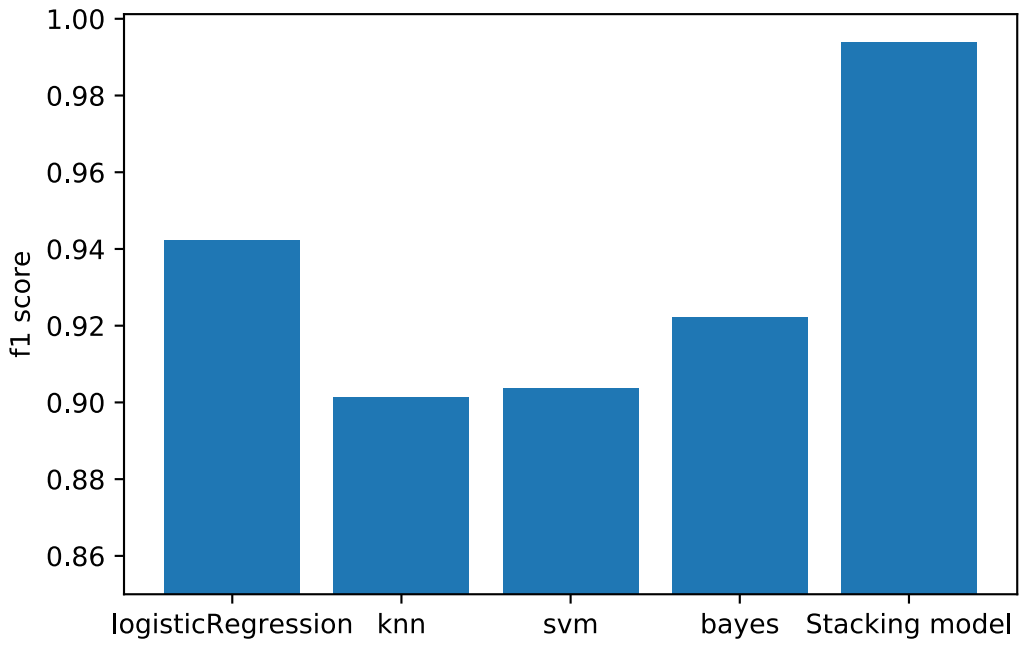

[17]:
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
```

```
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

base_learners = [
                 ('rf_1', RandomForestClassifier(n_estimators=40,␣
 ↪random_state=1)),
                 ('rf_2', KNeighborsClassifier(n_neighbors=10))
                ]

level0 = list()
level0.append(('lr', LogisticRegression(max_iter=500)))
level0.append(('knn', KNeighborsClassifier()))
level0.append(('cart', DecisionTreeClassifier()))
level0.append(('svm', SVC()))
level0.append(('bayes', GaussianNB()))

base_learners = level0
print(get_models())

# Initialize Stacking Classifier with the Meta Learner
clf = StackingClassifier(estimators=base_learners,␣
 ↪final_estimator=LogisticRegression(max_iter=500))
clf.fit(X_train, y_train)
clf_f1 = get_scores(clf)
```

```
{'logisticRegression': LogisticRegression(), 'knn': KNeighborsClassifier(),
'svm': SVC(), 'bayes': GaussianNB()}
accuracy:  0.9902574808629089
F1 score:  0.9940094137783483

Confusion matrix showing false/positives:
     0     1
0  523    11
1   17  2323
```

[18]:
```
results.append(clf_f1)
names.append("Stacking model")
samller_results = [x-0.85 for x in results]
plt.ylabel("f1 score")
plt.bar(names,samller_results,bottom = 0.85)
```

[18]:
```
<BarContainer object of 5 artists>
```




The f1 score is a better evaluation score to use than accuracy in this case, because the classes are uneven and predicting with a dummy classifierer will already score a accuracy of 80%. This chart shows the strength of a stacking model. Just using the logistic regression model has a f1 score of about 94%, whereas the stacking model, which uses logistic regression as a meta model and is trained with the data genereated by several base models has a f1 score of about 99%.

## 2 Bagging

For bagging the same data will be used.

[19]:
```
clf = BaggingClassifier(base_estimator=SVC(),n_estimators=10, random_state=0).
 ↪fit(X_train, y_train)
get_scores(clf)
```

```
accuracy:  0.8274182324286709
F1 score:  0.9038013964313422

Confusion matrix showing false/positives:
    0     1
0  48   486
1  10  2330
```

[19]:
```
0.9038013964313422
```

## 3 Boosting

[20]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
```

[21]:
```
gradient_boostclf = GradientBoostingClassifier(random_state=0)
gradient_boostclf.fit(X_train, y_train)
get_scores(gradient_boostclf)
```

```
accuracy:  0.9930410577592206
F1 score:  0.9957356076759062

Confusion matrix showing false/positives:
     0     1
0  519    15
1    5  2335
```

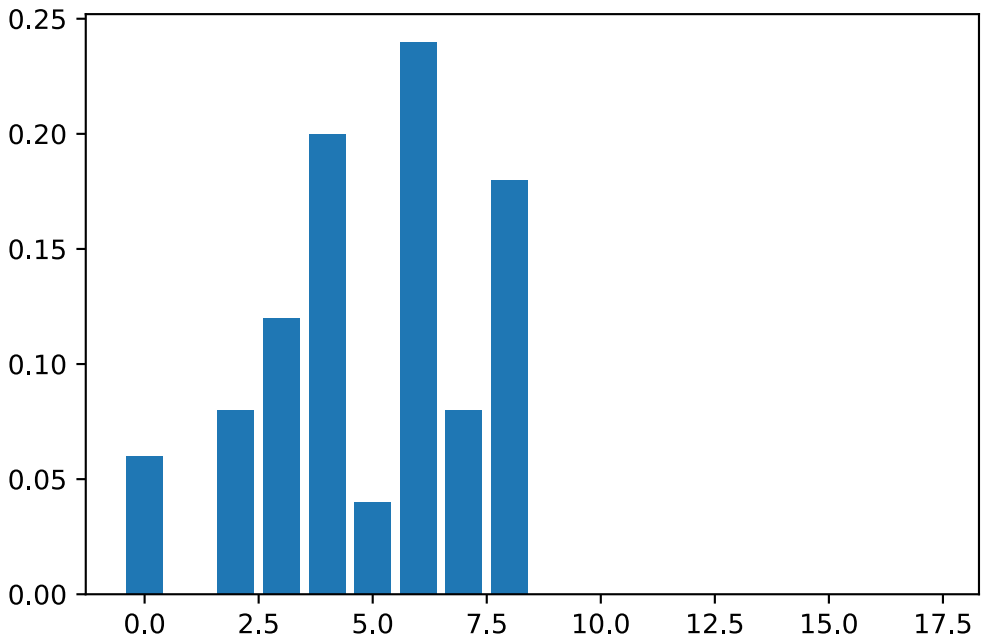
[21]:
```
0.9957356076759062
```

[22]:
```
ada_boostclf = AdaBoostClassifier(random_state=0)
ada_boostclf.fit(X_train, y_train)
get_scores(ada_boostclf)
```

```
accuracy:  0.9867780097425192
F1 score:  0.9918941979522183

Confusion matrix showing false/positives:
     0     1
0  511    23
1   15  2325
```

[22]:
```
0.9918941979522183
```

[23]:
```
feature_importence = ada_boostclf.feature_importances_
plt.bar(range(len(feature_importence)), feature_importence)
plt.show()
```



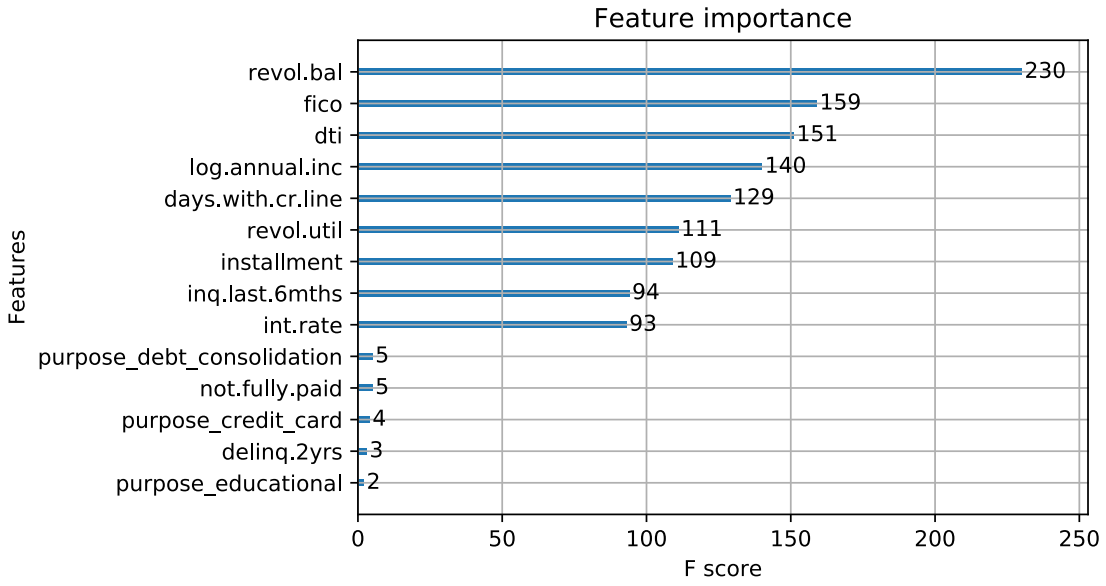

[24]:
```
from xgboost import XGBClassifier
from xgboost import plot_importance
```

[25]:
```
xgb_clf = XGBClassifier()
xgb_clf.fit(X_train, y_train)
get_scores(xgb_clf)
```

```
accuracy:  0.9951287404314544
F1 score:  0.9970111016225449

Confusion matrix showing false/positives:
     0     1
0  525     9
1    5  2335
```

[25]:
```
0.9970111016225449
```

[26]:
```
plot_importance(xgb_clf)
plt.show()
```




For the model that is performing excellent on this dataset the feature importance of this model shows, that the borrower's revolving balance is the most important feature for the decision of the credit policy. The next important features of making good predictions are the FICO credit score and the debt-to-income ratio of the borrower.

# 4 Automated Machine Learning (AutoML)

#### 4.0.1 AutoML is a technology to automate the building of predictave machine learning models.

[27]:
```
models_list = [ KNeighborsClassifier(),
                LogisticRegression(),
            StackingClassifier(estimators=base_learners),
            BaggingClassifier(base_estimator=SVC()),
            GradientBoostingClassifier(),
            XGBClassifier()

 ]


parameter_knn = {'n_neighbors': (range(2,50,5))}
parameter_logistic = {'solver':('newton-c', 'lbfgs', 'liblinear', 'sag', 'saga')}
parameter_stacking = {'final_estimator':(LogisticRegression(max_iter=500),␣
 ↪DecisionTreeClassifier())}
parameter_bagging = {'n_estimators': (range(5,45,20))}
parameter_gradientboost = {'max_depth':(2,4)}
parameter_XGboost = {'silent':([True])}
```

```
parameter_list = [
                    parameter_knn,
                    parameter_logistic,
                    parameter_stacking,
                    parameter_bagging,
                    parameter_gradientboost,
                    parameter_XGboost
]
```

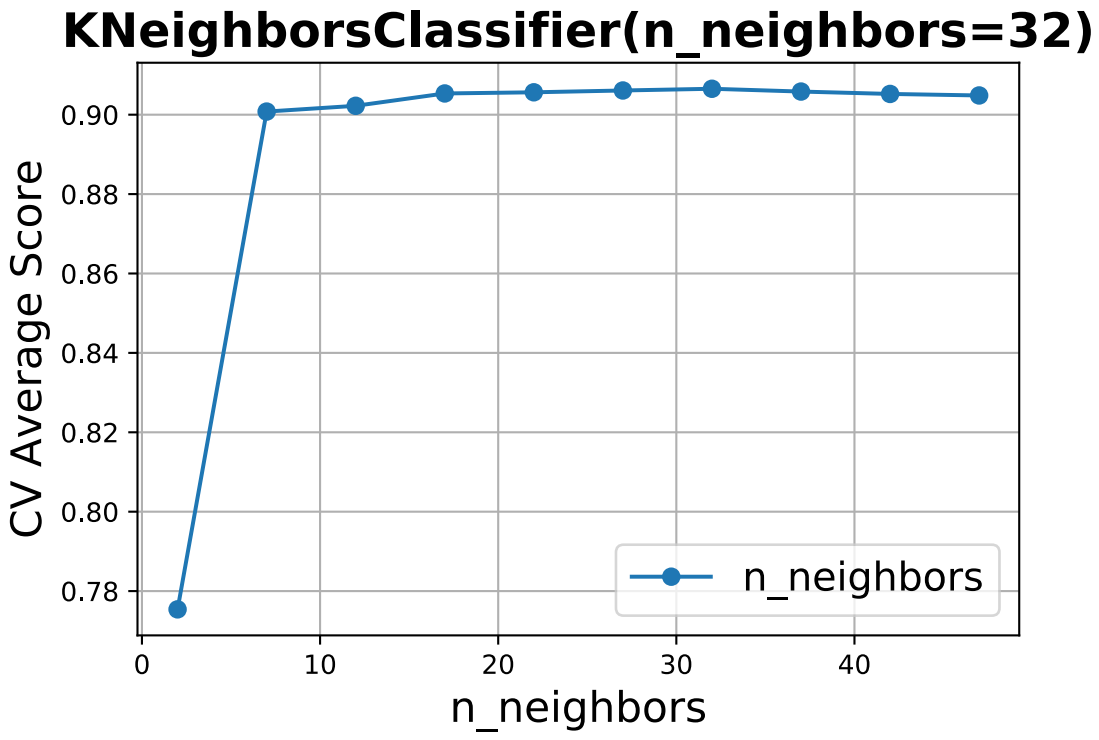
[28]:
```
def plot_grid_search(cv_results, grid_param_1, name_param_1):
    # Get Test Scores Mean and std for each grid search
    scores_mean = cv_results['mean_test_score']
    scores_sd = cv_results['std_test_score']

    # Plot Grid search scores
    _, ax = plt.subplots(1,1)

    # Param1 is the X-axis
    #check is the names for the x axis are string or number to print, if not␣
 ↪convert them
    if(isinstance(grid_param_1[0], str) or isinstance(grid_param_1[0], int) or␣
 ↪isinstance(grid_param_1[0], float)):
        ax.plot(grid_param_1, scores_mean, '-o', label= name_param_1)
    else:
        grid_params_str = list()
        for model_names_change in grid_param_1:
            grid_params_str.append(type(model_names_change).__name__)
        ax.plot(grid_params_str, scores_mean, '-o', label= name_param_1)

    ax.set_title(this_classifier.best_estimator_, fontsize=18, fontweight='bold')
    ax.set_xlabel(name_param_1, fontsize=16)
    ax.set_ylabel('CV Average Score', fontsize=16)
    ax.legend(loc="best", fontsize=15)
    ax.grid('on')
    #block=false to print directly
    plt.show(block = False)
```

[29]:
```
def train_get_score(grid_model,grid_parameters):
    print(grid_model)
    this_classifier = GridSearchCV(estimator = grid_model,param_grid =␣
 ↪grid_parameters,
                                   scoring="f1", n_jobs = -1, cv = 5)
    this_classifier.fit(X_train, y_train)
    best_estimator = this_classifier.best_estimator_
    print("Best estimator: ")
```

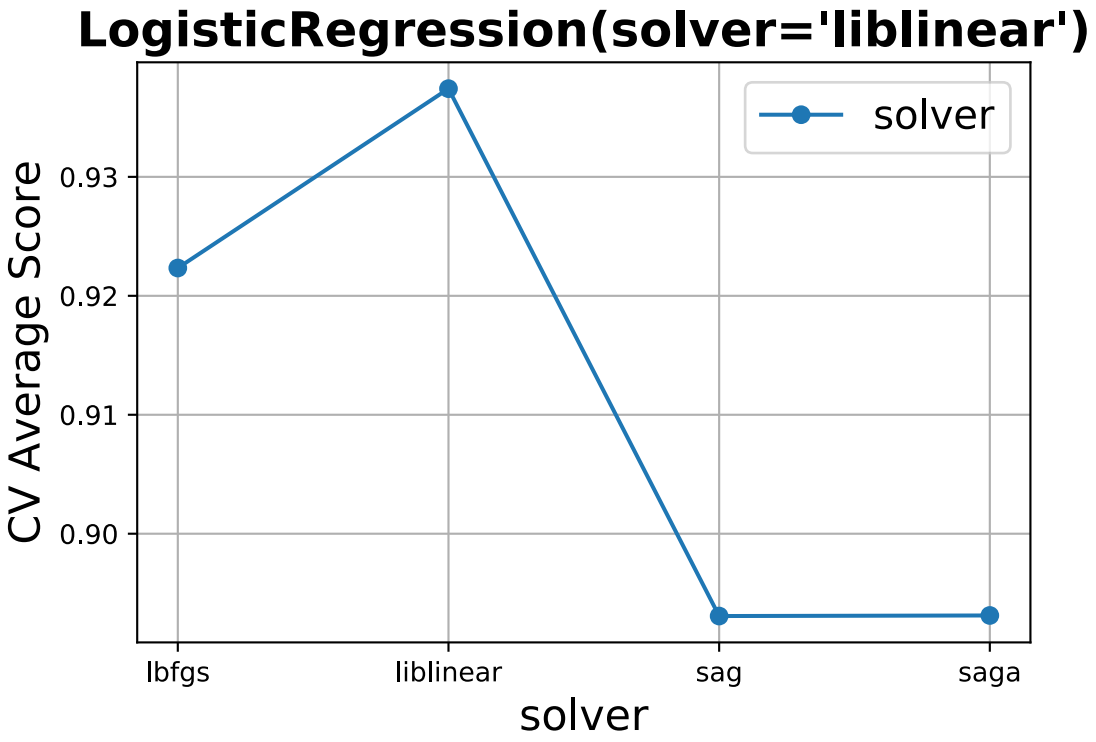
```
    print(best_estimator)
    this_score = just_get_scores(best_estimator)
    print(this_score)
    return this_classifier, this_score
```

[30]:
```
score_dict = dict()
for model_run,parameter_run in zip(models_list,parameter_list):
    this_classifier,this_score = train_get_score(model_run,parameter_run)
    score_dict[type(model_run).__name__] = this_score
    plot_grid_search(this_classifier.cv_results_, next(iter(parameter_run.
 ↪values())),
                         next(iter(parameter_run)))
```

```
KNeighborsClassifier()
Best estimator:
KNeighborsClassifier(n_neighbors=32)
0.9138473642800946
C:\Users\jonih\Anaconda3\envs\tf1\lib\site-
packages\matplotlib\cbook\__init__.py:424: MatplotlibDeprecationWarning:
Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated; use an
actual boolean (True/False) instead.
  warn_deprecated("2.2", "Passing one of 'on', 'true', 'off', 'false' as a "
```



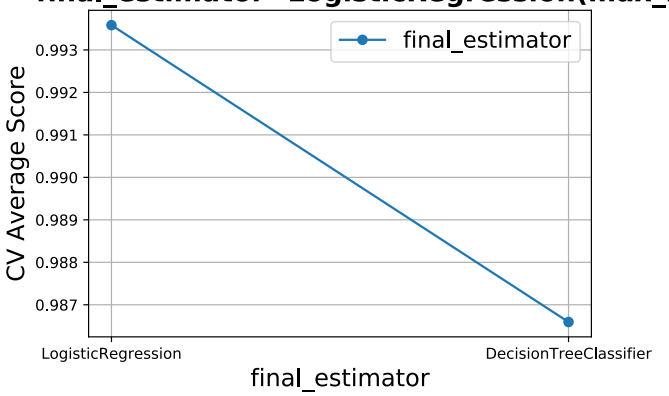
```
LogisticRegression()
Best estimator:
LogisticRegression(solver='liblinear')
0.9410548086866598
C:\Users\jonih\Anaconda3\envs\tf1\lib\site-
packages\matplotlib\cbook\__init__.py:424: MatplotlibDeprecationWarning:
Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated; use an
actual boolean (True/False) instead.
  warn_deprecated("2.2", "Passing one of 'on', 'true', 'off', 'false' as a "
```




```
StackingClassifier(estimators=[('lr', LogisticRegression(max_iter=500)),
                               ('knn', KNeighborsClassifier()),
                               ('cart', DecisionTreeClassifier()),
                               ('svm', SVC()), ('bayes', GaussianNB())])
Best estimator:
StackingClassifier(estimators=[('lr', LogisticRegression(max_iter=500)),
                               ('knn', KNeighborsClassifier()),
                               ('cart', DecisionTreeClassifier()),
                               ('svm', SVC()), ('bayes', GaussianNB())],
                   final_estimator=LogisticRegression(max_iter=500))
0.9935787671232876
C:\Users\jonih\Anaconda3\envs\tf1\lib\site-
packages\matplotlib\cbook\__init__.py:424: MatplotlibDeprecationWarning:
```

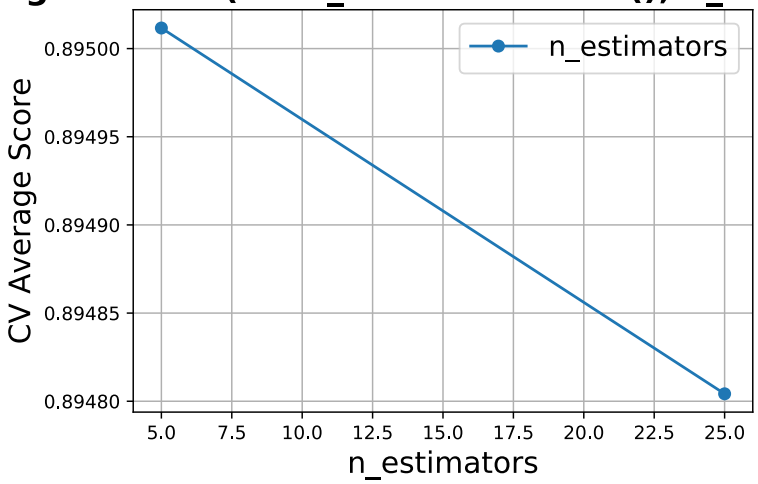
```
Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated; use an
actual boolean (True/False) instead.
  warn_deprecated("2.2", "Passing one of 'on', 'true', 'off', 'false' as a "
```

**StackingClassifier(estimators=[('lr', LogisticRegression(max_iter=500)),**
**('knn', KNeighborsClassifier()),**
**('cart', DecisionTreeClassifier()),**
**('svm', SVC()), ('bayes', GaussianNB())],**
**final_estimator=LogisticRegression(max_iter=500))**

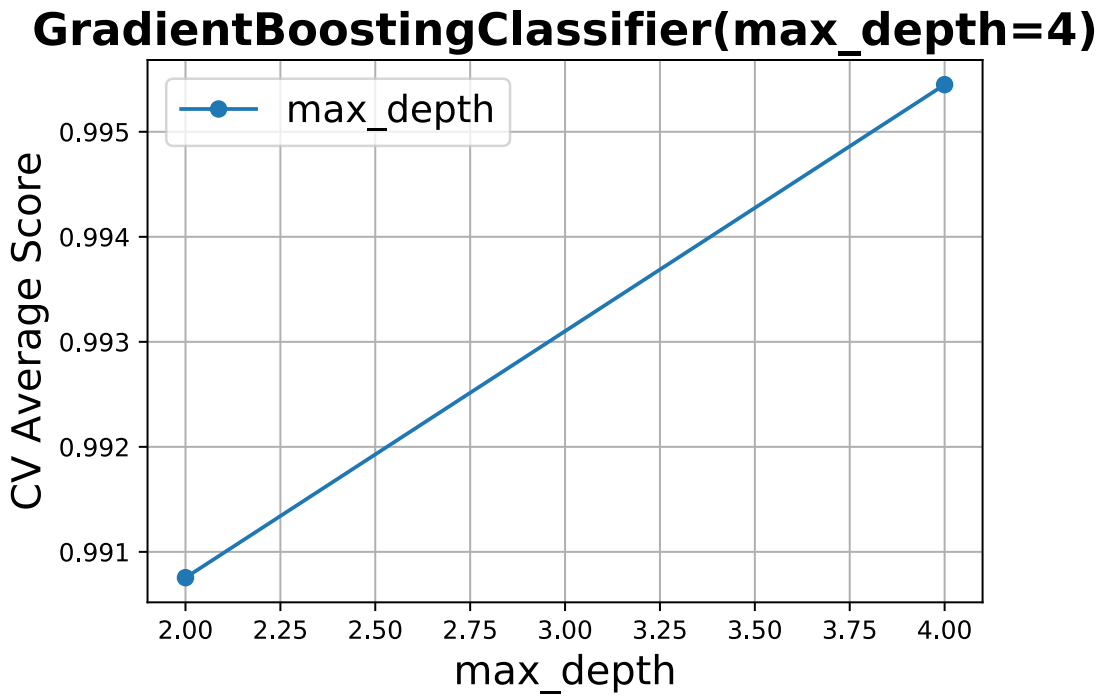



```
BaggingClassifier(base_estimator=SVC())
Best estimator:
BaggingClassifier(base_estimator=SVC(), n_estimators=5)
0.9038013964313422
C:\Users\jonih\Anaconda3\envs\tf1\lib\site-
packages\matplotlib\cbook\__init__.py:424: MatplotlibDeprecationWarning:
Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated; use an
actual boolean (True/False) instead.
  warn_deprecated("2.2", "Passing one of 'on', 'true', 'off', 'false' as a "
```

**BaggingClassifier(base_estimator=SVC(), n_estimators=5)**



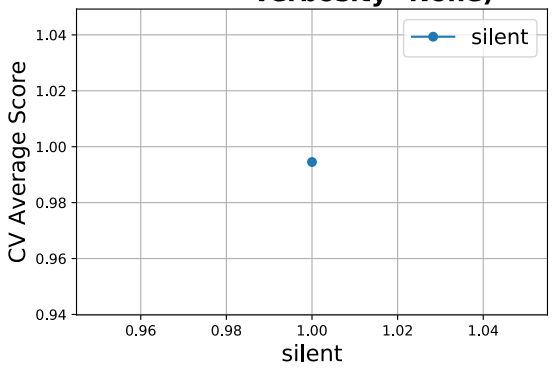
```
GradientBoostingClassifier()
Best estimator:
GradientBoostingClassifier(max_depth=4)
0.9965855740503629
C:\Users\jonih\Anaconda3\envs\tf1\lib\site-
packages\matplotlib\cbook\__init__.py:424: MatplotlibDeprecationWarning:
Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated; use an
actual boolean (True/False) instead.
  warn_deprecated("2.2", "Passing one of 'on', 'true', 'off', 'false' as a "
```




```
XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints=None,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=None, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
[10:34:25] WARNING: C:\Users\Administrator\workspace\xgboost-
win64_release_1.2.0\src\learner.cc:516:
Parameters: { silent } might not be used.
```

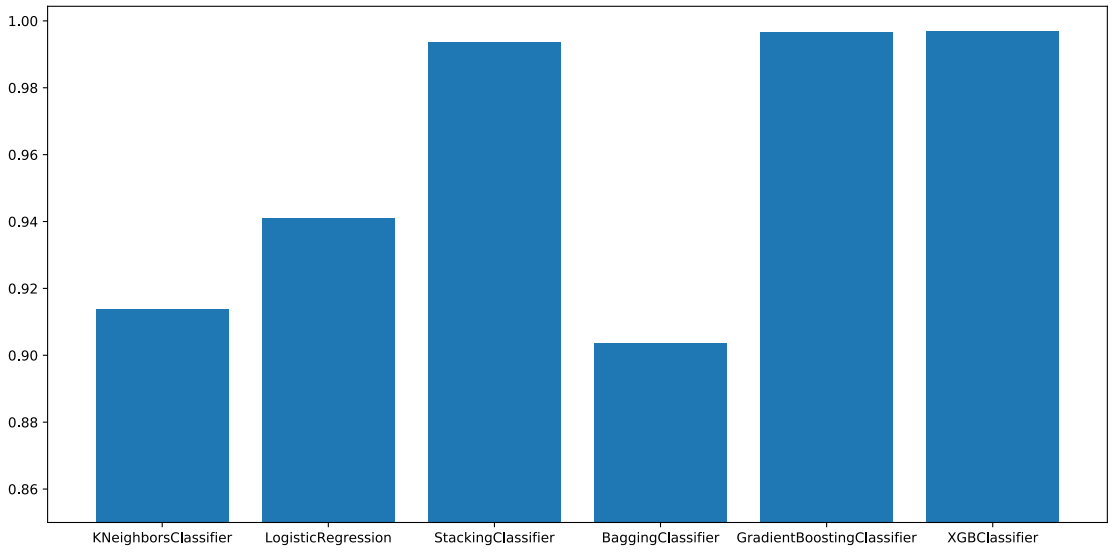
```
  This may not be accurate due to some parameters are only used in language
bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through
this
  verification. Please open an issue if you find above cases.


Best estimator:
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, silent=True,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)
0.9970111016225449
C:\Users\jonih\Anaconda3\envs\tf1\lib\site-
packages\matplotlib\cbook\__init__.py:424: MatplotlibDeprecationWarning:
Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated; use an
actual boolean (True/False) instead.
  warn_deprecated("2.2", "Passing one of 'on', 'true', 'off', 'false' as a "
```

**XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1, importance_type='gain', interaction_constraints='', learning_rate=0.300000012, max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, silent=True, subsample=1, tree_method='exact', validate_parameters=1, verbosity=None)**

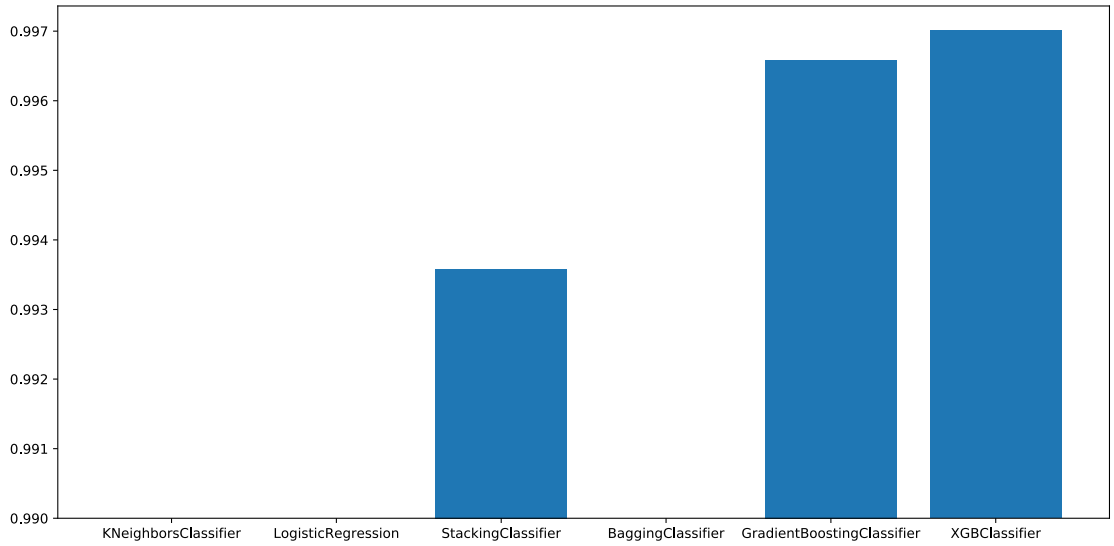


[40]:
```
best_classifier = max(score_dict, key=score_dict.get)
print("The best classifier for this prediction tast is "+ best_classifier)

#plot the accuracy of the models
keys = score_dict.keys()
values = score_dict.values()
samller_results = [x-0.85 for x in values]
plt.figure(figsize=(14,7))
plt.bar(keys, samller_results,bottom=0.85)
plt.show()

#f(x) if condition else g(x) for x in sequence
samllerer_results = [x-0.99 if ((x-0.99)>0) else x-x for x in values]
plt.figure(figsize=(14,7))
plt.bar(keys, samllerer_results,bottom=0.99)
```

```
The best classifier for this prediction tast is XGBClassifier
```




[40]:
```
<BarContainer object of 6 artists>
```




### 4.0.2 DEBBUGING STUFF maybe still needed

[188]:
```
this_classifier,this_score = train_get_score(models_list[5],parameter_list[5])
```

```
XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints=None,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=None, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
[16:51:07] WARNING: C:\Users\Administrator\workspace\xgboost-
win64_release_1.2.0\src\learner.cc:516:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language
bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through
this
  verification. Please open an issue if you find above cases.


Best estimator:
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
```

```
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, silent=True,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)
0.9970111016225449
```

[161]:
```
plot_grid_search(this_classifier.cv_results_,␣
↪next(iter(parameter_list[test_clf_nb].values())),␣
↪next(iter(parameter_list[test_clf_nb])))
```

```
<class 'str'>
['LogisticRegression', 'DecisionTreeClassifier']
C:\Users\jonih\Anaconda3\envs\tf1\lib\site-
packages\matplotlib\cbook\__init__.py:424: MatplotlibDeprecationWarning:
Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated; use an
actual boolean (True/False) instead.
  warn_deprecated("2.2", "Passing one of 'on', 'true', 'off', 'false' as a "
```

**StackingClassifier(estimators=[('lr', LogisticRegression(max_iter=500)),**
**('knn', KNeighborsClassifier()),**
**('cart', DecisionTreeClassifier()),**
**('svm', SVC()), ('bayes', GaussianNB())],**
**final_estimator=LogisticRegression(max_iter=500))**