K-fold Cross Validation

Abstract

In context of machine learning and statistics, data is the new currency, which is very costly to collect. Datasets that we obtain for predictive modelling are usually not as large as we want them to be to get a reliable model. That is why it is so important to use the data effectively, without wasting any single observation. This is where resampling method and cross-validation in particular are supposed to help us. In this assignment, I will investigate cross-validation and its application in machine learning.

1. Introduction and motivation

Model selection is usually done by dividing the data in two subsets: training and test subset, also known as validation subset. Each candidate model is fitted on the training set; then, each model's performance is evaluated with the validation set. After comparing performance metrics of the models, one can choose which one to use in production. The main issue with this approach is that we hold out a substantial part of the data (often 20-30%) for testing, which means this part of the data can never be used for model training. The reason for that is that a model can only be evaluated using "fresh" data, which it has never "seen" before; otherwise, we may face overfitting. Thus, when we have a scarce dataset, holding out a subset for testing becomes a problem, since reduction of training dataset is likely to reduce model's performance. However, seeking for more data is often not affordable: for example, if we work with customer churn data, collecting more observations would mean that we need more customers to leave, which obviously means losses for the company these customers left.

A common solution for this problem is applying a resampling method, such as cross-validation, which allows increasing model's performance and avoiding overfitting. Exploring cross-validation as an efficient way of data usage is the focus of this assignment.

The paper consists of a few paragraphs. In the next one, I introduce literature I familiarized myself with in order to prepare for this assignment and get ideas for my empirical analysis. The third chapter of this paper describes data and software I used. Then, in the fourth section of the paper, I formalize cross-validation and discuss its pros and cons. In the fifth section, I describe how I applied cross-validation for three purposes: model tuning, performance estimation and model selection; in addition, this paragraph includes a discussion of the results. The last paragraph summarizes this paper and shares my plans for future work.

2. Literature review

To understand the concept of cross validation, I familiarized myself with some scientific literature on related topics. In this section of the paper, I introduce my main findings.

The first paper I discovered was "Cross-Validation" by Daniel Berrar (Berrar, 2018). It provides a definition of cross-validation and an overview of validation types, such as random sub-sampling, Jackknife, k-fold and leave-one-out cross-validation. The paper also discusses their application in machine learning and provides ideas about which type of resampling methods should be used in practice.

Another important paper, also called "Cross-Validation", includes historical background and scientific fundamentals of the method (Refaeilzadeh, Tang & Liu, 2009). In addition, it explains more types of validation: hold-out validation, resubstitution validation, repeated k-fold cross-validation and others. This paper elaborates of why k=10 might be an optimal choice and mentions performance estimation, model selection and model tuning as the main applications of cross-validation.

One more paper I found helpful for my work was written by Herwig Friendl and Erwin Stampfer, also called "Cross-Validation". It explains the usefulness of cross-validation with flexible models, such as neural networks of tree-based classifiers, because these models are usually complicated and therefore easily lead to overfitting. This paper mentions special case of k-fold cross-validation, LOOCV (leave-one-out cross-validation), as well as some other resampling methods. One thing that also worth attention for this paper is that it describes environmental application of LOOCV with acid rain modelling.

Finally yet importantly, there is a "Train on Validation: Squeezing the Data Lemon" paper (Tennenholtz, Zahavy & Mannor, 2018). It introduces a method of using a validation set, which allows for a tradeoff between model's performance and overfitting. This method can be used in conjunction with k-fold cross-validation. The authors state that it can decrease bias while improving test performance (Tennenholtz et al, 2018).

3. Data description

This section of the assignment describes the dataset I used for applying cross-validation with the models (model description can be found is in section 5).

I found the data on Kaggle, a competition platform for programmers and data scientists. This is an open source data about bank customers churn, uploaded by Kaggle user Mehmet Akturk (Akturk, 2020). This data has tabular format and contains observations about customers of the bank. Each row contains information, such as customer ID, name, credit score, estimated salary, account balance, etc. Full list of features and their description is available on Kaggle (ibid). Each observation in the dataset is labeled with a binary target variable "Exited" identifying whether a customer left the bank (1) or stayed (0). It contains 10,000 rows and 12 columns (including the label) in total.

For simplicity, I will assume the same notation as in Daniel Berrar's paper (Berrar, 2018). Let us suppose that the dataset is denoted as D, and each observation is denoted as x_i, where $i = 1 \dots 10{,}000$. Since each observation x_i is labeled with classes, suppose true classes are denoted with y_i, where $i = 1, 0$, and predicted classes are denoted with $\hat{y}_i$, where $i = 1, 0$.

The data, when downloaded from Kaggle, was "raw", i.e. needed to be prepared before applying cross-validation or any of the models. I loaded it in as a csv file and took it through some cleaning procedures, such as data types change, imputation of non-available values, etc. More details on data preparation can be found in the code.

The software I used was Google Colab, and programming language was Python 3.7.12. The file with the code is uploaded with this paper; it is .ipynb format and can be read with e.g. Google Colab or Jupiter Lab.

4. Formalization of the method

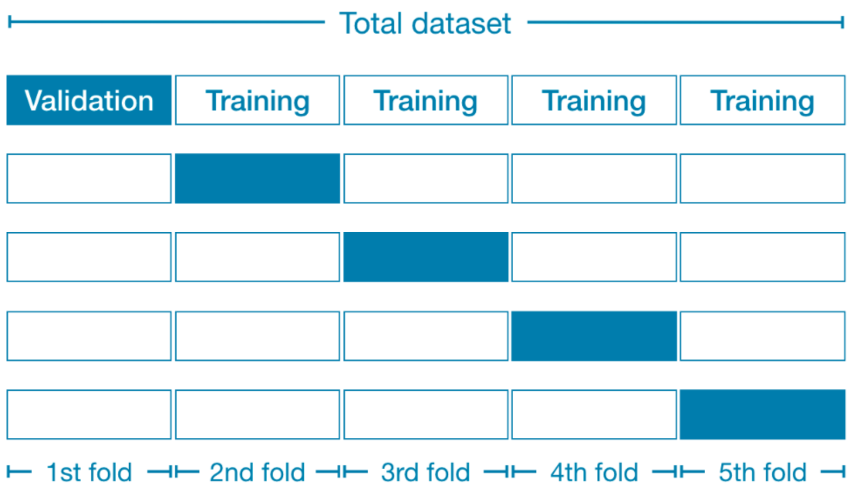
This section provides a general definition of cross-validation and gives a brief overview of validation types described in different literature. The focus of the paper, however, is k-fold cross validation, since this type is used the most widely. In addition, this part of the paper explains how k-fold cross-validation works and how it helps with model evaluation and selection. Furthermore, this paragraph includes discussion of pros and cons of the method.

I came across different definitions of cross-validation, and if I tried to consolidate them, I would formulate is as follows: cross-validation is a statistical resampling method for re-organizing the data for model evaluation and comparison. It is commonly applied in machine learning, because it is easy to use and helps to come up with a relatively unbiased measure for model performance. By re-organizing, they typically mean partitioning the data in a few subsamples for training and testing, instead of standard train-test split procedure, also known as hold-out validation (divide the data in just two parts: one for training the model and another for testing). Depending on goals of research, one can choose one of the existing types of validation.

One of them is, for example, resubstitution validation, a method, which implies training and then testing the model on the same data. Such an approach, however, is often considered a taboo in machine learning. Even though it allows using all data instances in model training, this validation process is a subject to overfitting. The model might perform very well on the existing data, but poorly on any future unseen data (Refaeilzadeh et al, 2008).

Another example method is k-fold random subsampling, which includes repeating a single hold-out k times, so that k pairs of training and test subsets are generated. Model is fitted to every training subset, and then it is applied to the test subset. Performance is estimated k times and then averaged over all k performance measures. In each pair, training and test subsets do not overlap; two training or two test sets, however, may overlap (Berrar, 2018).

There are many other validation methods, but some of the literature I came across with (e.g. Berrar, 2018) mentions, directly or indirectly, that none of these methods is as effective as k-fold cross validation. It is very similar to the above described random subsampling; the difference is that k-fold cross-validation does not allow two test sets to overlap. In k-fold cross-validation, the whole dataset is divided in k disjoint subsets: observations are randomly sampled from the entire data for it. The model is then trained on k-1 subsets and validated on the kth subset. Then, it is trained on other k-1 subsets and tested on the other kth subset, etc. This procedure is iterated until all the subsets are used for testing. As a result of testing of the model, we get k performance measures. A graphical illustration of this process with k = 5



Picture 1. Cross-validation with k=5. Source: https://www.kaggle.com/alexisbcook/cross-validation. Accessed on 16.02.2022

is shown on picture 1. An average over them is called cross-validated performance (ibid). One can use any metric for this; for example, mean squared error, accuracy, F-1 score, precision, recall, area under the ROC curve, etc. In my assignment, I will be using accuracy and area under the ROC curve. A detailed explanation of these measures can be found in the next paragraph.

A performance measure might be thought of as an error that a model makes when predicting a class. This error is calculated with loss function $L(y_i, \hat{y}_i)$. Prediction error can be then denoted as

$$\widehat{\varepsilon}_{CV} = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i),$$

where $\widehat{\varepsilon}_{CV}$ is the prediction error and n is the number of data instances (ibid).

A special case of cross-validation is a so-called leave-one-out cross validation, or LOOCV. Its main idea is that here $k = n$, i.e. the number of folds equals the number of data instances. Every individual data case is used for testing the model, which means the entire dataset except for one row is used for model training. In paper by Refaeilzadeh et al, it is mentioned that LOOCV yields high accuracy and low bias, but also high variance. This means bias-variance tradeoff cannot be satisfied with LOOCV (Refaeilzadeh et al, 2009).

Cross-validation is mainly applied in machine learning for performance estimation, model selection and model tuning. Performance estimation and model selection have been explained in sections 4 and 1, respectively. Model tuning, however has not been mentioned so far. When building a model, it is necessary to know which hyperparameters to set in order to achieve the best possible results with particular data. This is often called model tuning. How does one learn these parameters? It is a common solution to tune a model by means of cross-validation. Suppose we use the random forest algorithm. It has different hyperparameters to be tuned, for instance, maximum number of features to consider in each split, maximum depth of the tree, maximum number of trees, etc. There are different approaches to cross-validation tuning that are offered by scikit-learn, a free machine learning library for Python programming language (Benner, 2020). For example, grid search cross-validation, or GridSearchCV, by scikit-learn (sklearn.model_selection.GridSearchCV, 2021) which I used for model tuning for this assignment. When applying this algorithm, I set a grid of hyperparameters, which I wanted to optimize, with some values that I wanted the algorithm to try out for these hyperparameters. As an output of the code, I received a list with hyperparameter values to be used for model tuning. This is going to be explained more detailed in the next section.

Summarizing, one can claim that cross-validation is a widely used method, and it has many advantages. It provides a reliable measure of model performance; also, it uses as much data for training as possible, without “wasting” any data instance. In addition, it provides a few test errors (or a few performance measures) for every fold, which then can be averaged, and this gives a better idea about the range the actual model performance will likely be when it is put into production. All these advantages, however, come at a cost of high computational effort and a lot of time spent. This is a result of multiple iterations: the larger the k, the longer it gets to run the code (Kumar, 2019).

5. Empirical analysis

The initial idea of my empirical analysis was to try cross-validation for all three possible machine learning applications: performance estimation, model selection and model tuning. I also wanted to investigate how LOOCV fails to meet the bias-variance tradeoff comparing to 10-fold cross-validation, however, this was not possible due to limited computation power of my PC. Therefore, in this part of the paper, I will explain how I tried different applications of cross-validation with two models and what result I achieved.

5.1. CV for model tuning and performance estimation: XGBoost model

As already mentioned earlier, I chose bank customer churn data for my work. After finishing data preparation, I proceeded with setting up the benchmark model: gradient boosting based model. Gradient boosting is a machine learning algorithm, which is based on decision trees. It is an ensemble learning algorithm, which means it uses multiple models, for example, many decision trees. The idea of gradient boosting is combining “weak” functions, which are being built during an iterative process in which a new model is trained with respect to mistakes that the previous “weak” model made. Instead of creating a gradient boosting algorithm from scratch, I used a free library by scikit-learn: XGBoost (XGBoost Documentation, 2021). This library provides a gradient boosting classifier algorithm that can be used for binary classification. The goal of the model is to predict classes bank customers belong to: either leave the bank (1) or not leave (0).

I begin with model tuning. For doing that, I need to import two packages: XGBoost (ibid), and GridSearchCV from scikit-learn (GridSearchCV, 2021). GridSearchCV is a function with certain parameters. For simplicity, I did not use all of them, but chose those, which I thought were the most likely to influence the result. These are *estimator*, *param_grid*, *scoring* and *CV*. For more parameters, see GridSearchCV documentation (ibid).

Estimator object is to be chosen based on what problem the model is supposed to solve. As my problem is binary classification, I choose the XGBClassifier (a gradient boosting classifier). I decided to stick with it because this algorithm is a common choice for classification tasks like mine. This algorithm has proven to work well and won many Kaggle competitions. In addition, a large number of papers mentions that XGBoost algorithms yield the best performance. Some of these papers are “A Scalable Tree Boosting System” (2016) by Chen, T. & Guestrin, C.; “Comparison of Gradient Boosting Decision Tree Algorithms for CPU Performance” (2021) by Al-Shari, H., Saleh, Y.A. & Odabas, A.; and "An XGBoost risk model via feature selection and Bayesian hyper-parameter optimization” (2019) by Wang, Y. & Ni, X. S.

The next parameter of Grid Search cross-validation is *param_grid*, which stands for the grid of parameters for the model I am going to tune. GridSearchCV will go through this parameter grid and pick the best combination for this particular case. Again, there is a variety of parameters, but I chose just a few. Here they are:

- *colsample_bytree*: [0.1, 0.3] – this is the fraction of the training samples that will be used for each decision tree. The algorithm will go over different values from 0.1 to 0.3.
- *n_estimators*: [100, 400, 10] – this is the number of decision trees; the algorithm will look over values for this parameter from 100 to 400 with a step 10.
- *max_depth*: [3, 15] – maximum depth of a tree; values from 3 to 15 will be considered.
- *learning_rate*: [0. 1, 0.2, 0.05] – step size in every update to prevent overfitting.

More parameters can be found in documentation (XGBoost Documentation, 2021).

Another parameter of GridSearchCV was scoring. It means which method will be used for estimation of the model. I set it as ‘accuracy’.

The last parameter was CV, which is the number of folds to be used in cross-validation. As papers mention that k=10 is optimal (Berrar, 2018), I decided to stick with this value.

After setting parameters for GridSearchCV, I fit it to the training data. As a result, I get a set of parameters for my model tuning. See table 1.

Hyperparameter	**Value**
colsample_bytree	0.3
n_estimators	0.05
max_depth	3
learning_rate	400

Table 1. Optimal hyperparameters for model tuning

These parameters are optimal for this particular dataset and particular model. I use them in the next code cell for model tuning. Prior to applying tuned model, I import scikit learn metrics for assessing model performance: accuracy and area under the ROC curve (AUC).

Accuracy is a share of correctly classified cases; it shows how accurately an algorithm can predict classes. It is a common performance measure for binary classification tasks, which gives a relatively clear idea of model’s performance. Accuracy is calculated as follows:

$$Accuracy = \frac{TN+TP}{TN+FP+TP+FN},$$

where TN stands for “true negative”, i.e. quantity of data samples that were predicted to belong to a negative class and turned out to indeed be negative. Similarly, TP stands for “true positive”, FP – for “false positive”, and FN – for “false negative”.

AUC ROC is the area under the receiver operating characteristic curve. ROC Curve is a graphical visualization of the quality of a binary classification algorithm, as its discrimination threshold is varied. I chose this measure because it gives a clear graphical image, which is convenient to compare two models (Fawcett, 2006). AUC ROC can be thought of as a numerical equivalent of the graph with the ROC curve.

I assigned the tuned model a name “tuned_xgb” and plugged the obtained parameters in the model. After that, it was time to fit the model on training data.

The model was first validated on the test set of size 30% of the data without cross-validation, since I wanted to see how the results would differ if I used cross-validation for performance measurement. I used functions *accuracy_score()* and *roc_auc_score()* for that. Then, I checked performance measure by means of cross-validation for each of the 10 folds. This is a simple procedure if one uses an existing package from scikit-learn library. I imported function *cross_validate(),* created a list of measures I wanted to use (accuracy, ROC AUC) and used this list as one of the input parameters for *cross_validate()* function. The results can be observed in table 2.

Measure	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Fold 7**	**Fold 8**	**Fold 9**	**Fold 10**
Test accuracy	0,86	0,87	0,84	0,88	0,89	0,85	0,89	0,88	0,87	0,87
Test ROC AUC	0,90	0,88	0,83	0,86	0,91	0,87	0,91	0,90	0,87	0,87

Table 2. Test AUC and accuracy of XGBoost model measured for 10 folds

Each fold's score in AUC and accuracy lies in the interval between 0.80 and 0.91. Both ROC AUC and accuracy scores are considered good if they are close to one. Thus, XGBoost model's performance turns out to be a reasonably good result.

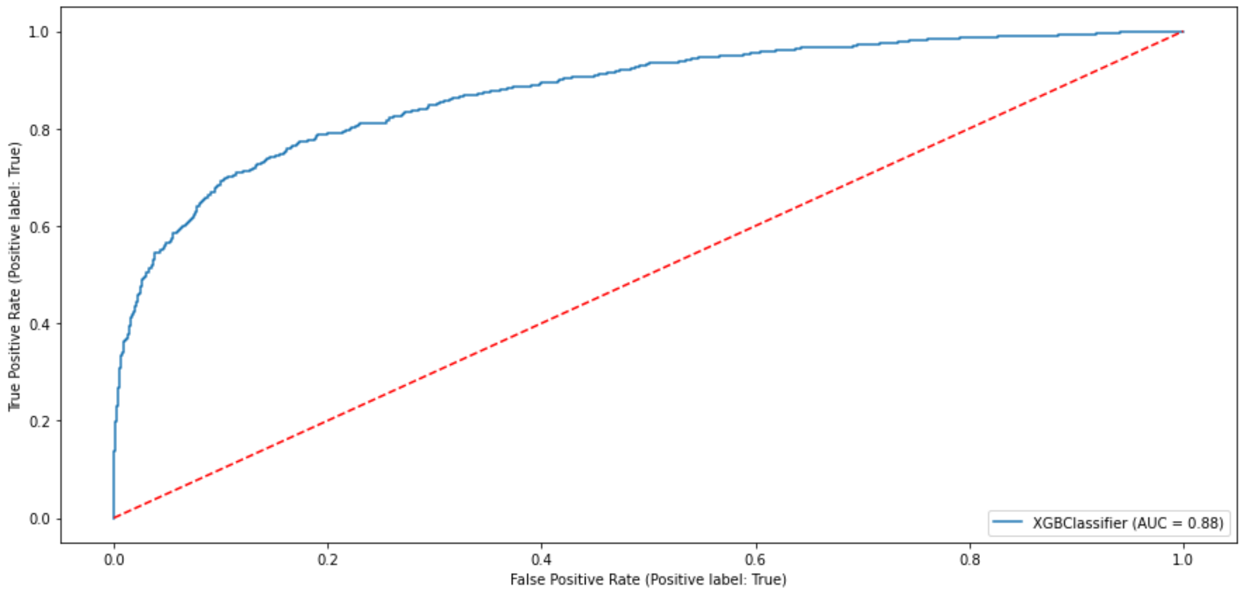
Then, in order to compare non-CV performance measures with those obtained with cross-validation, I averaged over all of the ten metrics using function *mean()* and converted the result to a pandas data frame. See table 3.

Measure	**Non-CV performance**	**Cross-validated performance**
Test accuracy	0.870667	0.870667
Test ROC AUC	0.879094	0.879843

Table 3. Comparison of accuracy and AUC scores with and without cross-validation

As it can be observed, there is a minor difference between the scores, which, I guess, is something I should have expected; cross-validation is supposed to help obtaining a less biased measure of performance, but not something totally different.

In addition to the scores, I made a graphical illustration: ROC curve. It can be found on picture 2.



Picture 2. ROC curve of XGB Classifier

This is to illustrate what ROC AUC score is showing: the area under the blue line. On the vertical axes, there is true positive rate (the proportion of correct predictions in predictions of positive class), and on the horizontal axes is the false positive rate (the proportion of false predictions in predictions of positive class). Red line represents a random benchmark, i.e. what the curve would look like if the algorithm "guessed" the classes randomly. The closer the blue line to the upper left corner of the graph is, the better job the algorithm does.

5.2. CV for model selection: Random Forest model vs XGBoost model

The previous sub-chapter describes application of cross-validation for model tuning and performance estimation. In this sub-chapter, I also show how I used cross-validation for model selection. A worthy opponent to XGBoost model is Random Forest. As one could guess from the name, this is also a tree-based ensemble learning method for classification and regression. It uses a multitude of decision trees in randomly selected sub-spaces of the feature space. When the task is classification, the output of the algorithm is a prediction of a class label each data instance belongs to, for which "voted" the majority of the decision trees (Ho, 1995).

As previously, I used a free package from scikit-learn library. I imported Random Forest classifier and started with Grid Search Cross-validation to find the best hyperparameter combination for model tuning. Again, I had to choose which parameters to set inside of the GridSearchCV function. Since I am using another classification algorithm, I have to choose a different *estimator*: RandomForestClassifier. The *CV* and the *scoring* parameters remain the same: k=10 and 'accuracy', respectively. Another distinction with the previous model tuning is the set of parameters GridSearchCV algorithm will select for the model. The new set of parameters is:

- *n_estimators*: [100, 400, 10],
- *max_depth*: [3, 15],
- *criterion*: ["gini", "entropy"].

The new input parameter is *criterion*. It is responsible for measurement of the quality of a split on the features. In scikit-learn library, possible option is either "gini" or "entropy". "Gini" stands for the gini impurity. It measures the frequency at which any row of the dataset is assigned a wrong class label when it is labeled randomly. It is calculated using this formula:

$$Gini\ Index = 1 - \sum_{i=1}^{n} p_i^2,$$

where p_i is the probability of a data instance belonging to a class i. Entropy reflects information about disorder between features and the target variable. It is calculated with the formula:

$$Entropy = -\sum_{i=1}^{n} p_i \log_2 p_i.$$

Gini has to stay within interval $[0, 0.5]$, and entropy – within interval $[0, 1]$. The algorithm of grid search cross-validation will need to decide, which of the two criteria will result in a higher model performance given the dataset.

Now as all the new inputs are chosen, the code is launched. Its output is again a list of suitable parameter values. It can be observed in table 4.

Hyperparameter	**Value**
criterion	'gini'
max_depth	15
n_estimators	400

Table 4. Optimal hyperparameters for model tuning

These values are then plugged in the tuned model. I called it "tuned_forest" and used these parameters as an input for function RandomForestClassifier(), which is available from scikit-learn library. After that, I fit the model on the train data and obtain predictions on the test set for the class labels. As previously, I check the measures of performance with no cross-validation and get accuracy score of 0.81 and AUC score of 0.84. Then, with help of *cross_validate()* function, I run 10 iterations of model validation and obtain 10 scores for both accuracy and AUC metrics. Results are available in table 5.

Measure	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10
Test accuracy	0,82	0,82	0,81	0,82	0,82	0,80	0,84	0,83	0,83	0,81
Test ROC AUC	0,86	0,85	0,77	0,85	0,87	0,81	0,87	0,86	0,82	0,85

Table 5. Test AUC and accuracy of Random Forest model measured for 10 folds

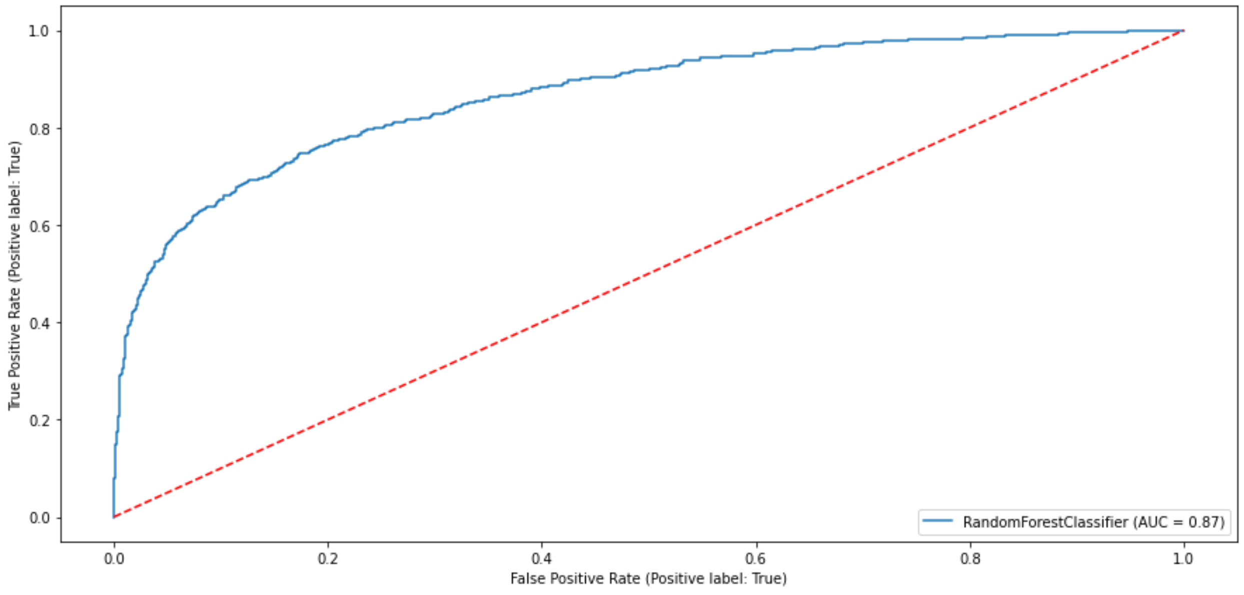
Again, with a *mean*() function I average over these measures and get a cross-validated accuracy and AUC. Results are in table 6.

Measure	Non-CV performance	Cross-validated performance
Test accuracy	0.833333	0.819000
Test ROC AUC	0.836706	0.841091

Table 6. Comparison of accuracy and AUC scores with and without cross-validation

Just like in case with XGBoost model, there is a minor difference in scores when measured on a 30%-sized test subset and by cross-validated performance measures. Cross-validated test accuracy is lower than “non-CV” accuracy, and AUC score is higher in case of cross-validated AUC score. With cross-validated performance measures, I can be more confident about correctness of model estimation I get.

Similarly to XGBoost model, I made a graphical representation of AUC score of Fandom Forest model. This is done with a function *plot_roc_curve()* from metrics by scikit-learn and with matplotlib library. See picture 3.



Picture 3. ROC curve of Random Forest Classifier

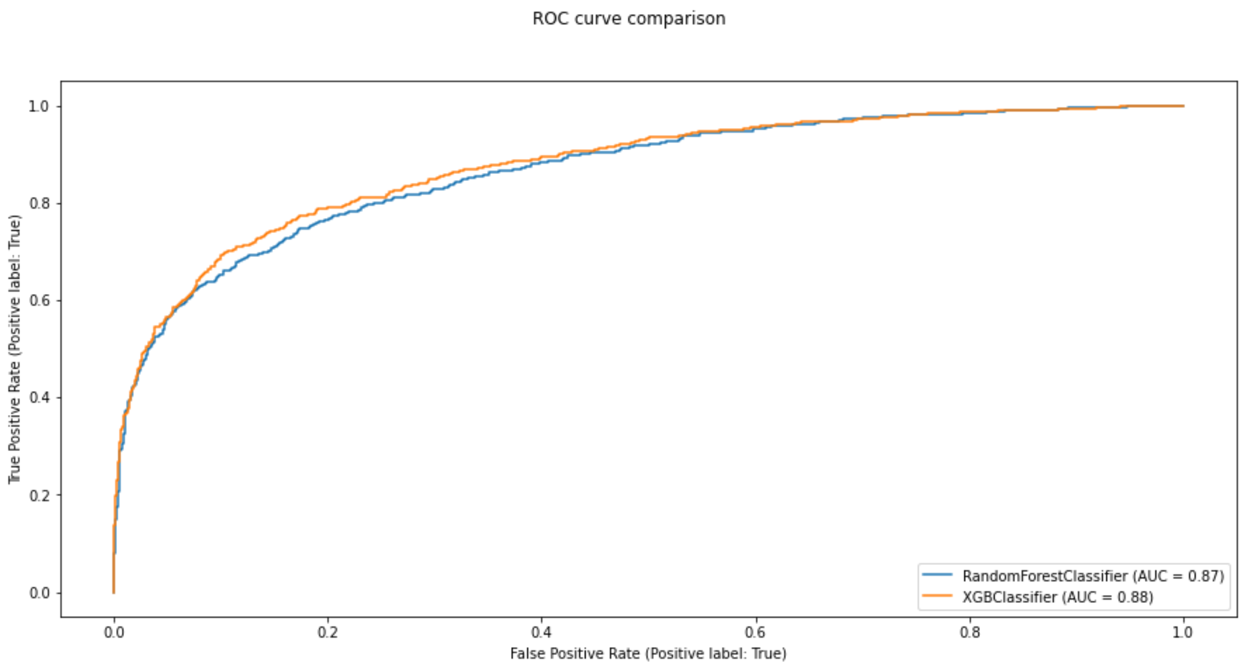
It is observable that Random Forest model’s performance is very close to the benchmark XGBoost model. Random Forest performed slightly worse. In table 7, I gathered cross-validated performance measures of both of the models.

Cross-validated performance measure	**XGBoost model**	**Random Forest model**
Accuracy	0.870667	0.819000
ROC AUC	0.879843	0.841091

Table 7. Comparison of cross-validated performance measures of XGBoost and Random Forest models

With these cross-validated performance measures, I can select a model that worked out better. Clearly, if I were to put one of these models in production, I would choose XGBoost model. 10-fold cross-validation helped me with this task.

As a final note, I include a graphical representation of both ROC curves: picture 4.



Picture 4. ROC curves of XGBoost and Random Forest classifiers

This graph makes a clear visual difference between the qualities of the two algorithms and confirms the results of cross-validation performance measurements. The yellow line is the XGBClassifier, and it is slightly closer to the upper left corner and therefore can be consider a better classifier.

6. Summary and conclusion

In conclusion, I can say that k-fold cross-validation is indeed an easy to use method that is extremely helpful with at least three machine learning applications: model tuning, performance estimation and model selection. I made sure this is true when I tried it by myself; no wonder it is one of the most widely used methods.

To summarize this work, recall the most important points about cross-validation. K-fold cross-validation is a resampling method that re-organizes the data. The entire dataset is divided into k disjoint folds and a model is trained on k-1 folds and tested on the kth fold. This procedure is iterated until there is no subset left which has not been involved in validating the model. As a result, k performance measures are available. An average over these k measures is called cross-validated performance measure. Cross-validation is helpful when the amount of data is limited to use the data in the most efficient way; however, this method is time-consuming and costly in terms of high computational effort.

Using some available packages from scikit-learn library, I managed to explore cross-validation myself for model tuning with grid search cross validation, performance estimation with cross-validated accuracy and AUC, and model selection with the same metrics.

For the future work, I want to try leave-one-out cross-validation, when I have hardware with sufficient computing power. It is very interesting for me to measure how bias and variance differ when either 10-fold cross-validation or LOOCV is applied.

7. References

Data and pictures:

1) Akturk, M. (2020, July 25) Churn for Bank Customers. Predict customer churn in a bank. *Kaggle*. Retrieved from https://www.kaggle.com/mathchi/churn-for-bank-customers. Accessed on 08.01.2022
2) Cook, A. (2021, November 09) Exercise: Cross-Validation. *Kaggle*. Retrieved from https://www.kaggle.com/alexisbcook/exercise-cross-validation. Accessed on 10.02.2022

Literature:

1) Al-Shari, H., Saleh, Y.A. & Odabas, A. (2021). Comparison of Gradient Boosting Decision Tree Algorithms for CPU Performance. *Erciyes Tip Dergisi*, 157-168
2) Benner, J. (2020, August 06) Cross-Validation and Hyperparameter Tuning: How to Optimize your Machine Learning Model. *Towards Data Science*. Retrieved from https://towardsdatascience.com/cross-validation-and-hyperparameter-tuning-how-to-optimise-your-machine-learning-model-13f005af9d7d. Accessed on 15.02.2022
3) Berrar, D. (2018). Cross-Validation. *Encyclopedia of Bioinformatics and Computational Biology, 1*, 542-545. DOI: 10.1016/B978-0-12-809633-8.20349-X
4) Chen, T. & Guestrin, C. (2016). A Scalable Tree Boosting System. *In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794
5) Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27, 861-874. DOI: 10.1016/j.patrec.2005.10.010
6) Ho, T.K. (1995). Random Decision Forests. *Proceedings of the Third International Conference on Document Analysis and Recognition*, 278–282.
7) Kumar, N. (2019, February 27) Advantages and Disadvantages of Cross Validation in Machine Learning. *The Professionals Point*. Retrieved from http://theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of-cross.html. Accessed on 15.02.2022
8) Refaeilzadeh, P., Tang, L. & Liu, H. (2009). Cross-Validation. *Encyclopedia of Database Systems.* DOI: 532–538. 532-538. 10.1007/978-0-387-39940-9_565
9) sklearn.model_selection.GridSearchCV. (2021) sklearn.model_selection.GridSearchCV documentation. *Scikit-learn*. Retrieved from: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Accessed on 18.02.2022
10) Tennenholtz, G., Zahavy, T. & Mannor, S. (2018). Train on Validation: Squeezing the Data Lemon. *ArXiv, abs/1802.05846*.
11) Wang, Y. & Ni, X.S. (2019). An XGBoost risk model via feature selection and Bayesian hyper-parameter optimization. *International Journal of Database Management Systems*, 11. DOI: 01-17. 10.5121/ijdms.2019.11101
12) XGBoost Documentation (20201) XGBoost Documentation. *Readthedocs*. Retrieved from https://xgboost.readthedocs.io/en/stable/. Accessed on 18.02.2022