Group Project – Summary

In our group project, we had to build an algorithm that produces price class estimations (division into 5 classes) based on a dataset containing data about residential homes in Ames, Iowa. To achieve that, we built a model with the use of neural nets and compared this approach to decision trees and random forests as well as boosted forests. We found out that the neural net approach wasn't the most appropriate for this task since the dataset is rather small and neural nets unlock their full potential when they have tons of data to train on. When comparing the different approaches, boosted forests tended to perform best, followed by random forests. Neural nets and decision trees performed a bit worse. But overall, one can say that all models performed well when looking at the confusion matrices (see at the end of the Jupyter Notebook). There was little misclassification and if there was, then often only by one class.

1. Handling of the data

The first thing we had to do was the handling of the data we were given. This OpenML dataset contained 79 explanatory variables about quite a lot of aspects of residential homes, so we selected features which we think have an impact on the price. In a second step, we prepared the data in a way we could feed to the models.

1.1 Feature selection

In order to select the features, we looked at the dataset and used our intuition as to what may have a significant impact on the price of a house. Some of our group's members study Banking and Finance and know what to look out for when pricing real estate. We noticed that some features weren't really helpful to make distinctions because they were almost the same for all houses (e.g. "Heating" was "GasA" for almost all entries). Other features were correlated with each other (e.g. "FireplacesQu" and "Fireplaces", which also makes sense because the quality of a fireplace is NaN when there isn't any), so we decided to just keep one of them. There were also features like "PoolArea" and "PoolQC" (standing for pool quality), which for sure have an impact on the price of a house but were just too rare all over the dataset because there were simply very few homes with pools. Because a machine learning algorithm won't learn much from such datapoints, we decided to exclude features like this.

1.2 Handling of categorical and numerical data

For dealing with categorical data, we used the pandas get_dummies() function to make sure that the models can handle the data correctly. That function splits one feature into multiple features with one-hot encoding.

A lot of the features that recorded some kind of quality (e.g. "BsmtQual" or "KitchenQual") were given in strings (usually "Ex" for "excellent", "Gd" for "good", "TA" for "average/typical", "Fa" for "fair", "Po" for "poor" and then also "NA", more on the handling of that in 1.2.2). We then first converted these strings into numbers (e.g. "good" is always better than "poor", therefore we can represent it as numbers) and scaled them between zero and two for all numerical features across the dataset.

1.2.1 Composite features

Out of three features that only described dates on which certain things happened (to be specific, "YrSold", "YearBuilt", and "YearRemodAdd") we built two composite features. The idea behind that was that the dates alone don't have much explanatory power, but if we connect them with each other, we can get more significant features. We came up with the following two features:

- “AgeWhenSold”: difference between the date the house was sold (“YrSold”) and the date the house was built (“YearBuilt”)
- “AgeSinceRemod”: difference between the date the house was sold (“YrSold”) and the date the house was last remodeled (“YearRemodAdd”)

The computation of the feature importance for the random forest model at the end of our Jupyter Notebook shows that our composite features are in the top-7 of the most significant features for that particular model. Note that the feature importance strongly depends on the seed one sets for the shuffling of the data. Nevertheless, this result suggests that our composite features are meaningful for the model.

1.2.2 Dealing with NaN values

When reading through the data description we quickly figured out that NaN values don’t necessarily mean that there’s no data available but more often they mean that certain features are not present in a house (e.g. in “FireplaceQu” “NA” means that there is no fireplace, in “GarageType” it means that there is no garage, etc.). That’s why it would be a mistake to just entirely kick them out of the dataset (that was done in the minimal working example with columns that contained more than 40% NaN values). Considering the data description, we figured out that there were in fact no “real” NaN values within our chosen subset of data.

2. Data imbalance

Data imbalance was a real problem with the given dataset. Already when looking at the minimum working example (in particular the histogram as output of the 5th box of code) one can see that there are a lot of datapoints for the second class while there are barely any houses belonging to the 5th class.

In order to tackle this problem, we did a little experiment and fused classes 4 and 5. We found that it improves testing accuracy, but we also would have to take into account that since there’s no longer a 5th class the testing accuracy on class 5 is zero. One would have to weigh whether this would be worth it for the sake of a higher overall testing accuracy or whether it’s more important that the model makes estimations about all the classes. We decided that for this task we should get 5 classes as output, so we left classes 4 and 5 as they were.

We also thought about oversampling (duplicate entries for classes with less datapoints), but this would result in an overfit and the algorithm would just learn the datapoints by heart. There would also be the possibility to remove entries from classes with a lot of datapoints so that they match the one with the least (in our case class 5), but this would result in too little data left for building a model. It’s important to keep in mind that when handling the imbalance only the training data should be balanced, so the test data is independent.

2.1 Note: Shuffling of data

We found it to be best (and easiest) to randomly shuffle the entire dataset before splitting it into training and testing data. This eliminates any kind of order or structure the dataset may have had, making cross-validation less of a concern. For the neural net we additionally had to set aside some validation data, but that was not necessary for the other models (decision trees, random forest, boosted forests) since the scikit-learn library handles that automatically.

The shuffling makes a huge difference for the testing accuracy. We experimented with different seeds for the shuffling. Interestingly, a seed of 67 resulted in good testing accuracy while a seed of 4 caused a rather poor testing accuracy. This tells us that we do not have a large enough dataset.

But why is the testing accuracy different depending on the shuffling and splitting of the data? A different seed will produce a different order of datapoints. Since we then take the first 80% of the data for training and the last 20% for testing, we always end up with different testing and training sets. The model is trained and tested on different splits each time we change the seed.

2.2 Tomek Links

Before we trained our models with the training data, we applied a technique called Tomek Links to our training data. The idea of this technique is to remove datapoints which are very close to each other but belong to different classes (Tomek (1976)). This way it’s easier for the classification algorithm to draw a boundary between the different classes. It seemed to us like this procedure balanced the data slightly and had a small but positive effect on testing accuracy.

3. Models

3.1 Neural Net

Inspired by discussions in class, we decided to try neural nets. Neural nets are part of deep learning and supervised machine learning and follow the simple structure of trial and error in order to minimize the loss-error (Mesquita (2021)).

In our Jupyter Notebook, we very briefly explained the approach of neural nets based on Nielsen (2015) and Chollet (2017). It’s important to mention that neural nets are very powerful and tend to learn all the datapoints by heart, especially when the dataset isn’t that big like in our setting. To solve this problem of overfitting the neural net must be stopped in the learning process at some point, otherwise the testing/validation accuracy will slowly start to decrease again.

When working with neural nets, there are tons of parameters one can vary (number of layers, activation functions in each layer, optimizer, loss functions, etc.). For this exercise, we went with settings we think are good but didn’t have the time and resources to go into hyperparameter optimization (it is computationally intensive to do so).

3.2 Decision Trees

Decision Trees are a very popular approach as it is in general very intuitive and therefore easy to interpret. By posing a sequence of questions we can close in on the class labels.

In order to draw up/plot the decision tree we need to split the space of possible values (X_1 to X_p) into optimally M non-overlapping sectors (R_1 up to R_M). After the Division into those regions, we need to model the response as constant for every observation in R_m. In our case the number of splits has been limited to 10, 12, 15, 17 and 20. To combat impurity in the process of formulating the decision tree, we implemented a fixated range for the two impurity measures “Gini” and “Entropy”. It must be mentioned here that the Gini Index and the Entropy approach tend to produce quite similar results in practice. All in all, the method of creating decision trees is a classification approach but there are some disadvantages that come with it. For one there seems to be an instability of the trees when we analyze the result of small data changes which often lead to a very different series of splits. (Breiman (1984))

Another thing to keep in mind is the tendency to rather complex decision boundaries which regularly is accompanied by overfitting. Lastly, decision trees are not the best choice regarding their predictive accuracy when compared with other approaches we discuss.

3.3 Random Forest

The following text is mainly based on the empirical findings and the general overview provided by Breiman (2001) in his paper at the Statistics Department of the University of California. Random Forest combines multiple decision trees to improve the accuracy and reduce the variance of the model. Given that it has been labeled as an “ensemble learning” algorithm. These algorithms produce k independent predictions which then assign the class label based on a majority vote of the k outcomes.

The result of our Random Forest approach lays in the range of the testing accuracies of 0.81 to 0.86. In general, the Random Forest method is regarded as rather robust contrary to using only 1 decision tree as above. The literature also supports this fact and was able to show the main advantages and disadvantages in using this approach via the paper “Random Forests for Classification in Ecology”.

Some of the following remarks are also based on Cutler et. al. (2007). It also usually tends to be more accurate than the decision tree approach. However random forest poses some problems regarding the complexity. It is rather slow to train on large datasets with many features and can be computationally rather expensive, all depending on the number of trees set of course. Overall, it is a powerful algorithm, but it is essential to tune the parameters carefully to achieve the best results and to keep its limitations in mind.

3.4 Boosted Forests

When we talk about the idea of “boosting” an algorithm, it quite simply means that we combine small decision trees with each new tree trying to correct the errors of the previously created tree. Therefore, the main difference to the classical random forest approach is that here the trees are not created independently from others. The following remarks are based on Choudhury (2021).

We used two different boosting approaches in our project, on one hand the Gradient Boost and on the other hand the AdaBoost. As a boosted model trains and predicts on a subset of data, it reduces memory requirements and therefore the computation time. Combined with the provided measure of feature importance from this method and the resulting better understanding of the data patterns, boosted forests seem to be a very good approach to use. However, it is imperative to tune the hyperparameters in a careful manner to achieve the sought after optimal performance, which can be time consuming. Another disadvantage of boosted forests is the complexity which often hinders the interpretability of the models results.

The AdaBoost model combines the predictions of weak classifiers. AdaBoost focuses more on instances that are hard to classify, thus improving the overall accuracy of the algorithm. Gradient Boost works by iteratively adding weak classifiers to a classification model, where each subsequent classifier is fitted to correct the errors of the previous classifier. The model focuses on the residuals of the previous model. The next model then fits the residuals of the previous model. The final prediction is made by combining the predictions of all weak classifiers, with each classifier's weight determined by its contribution to the reduction of the overall error. Gradient Boost is most effective in handling continuous data.

3.5 Model comparison

When comparing Neural Networks to Decision Trees, we find that both models can perform equally well. If the Neural Network model performs poorly, then the Decision Tree model will also perform poorly. However, the boosted models, which iteratively minimize the error, tend to perform the best. This is also visible in the confusion matrices, where there are fewer drastic misclassifications, and if there are, they are usually off by only one class (see Jupyter Notebook for visualization). Boosted models are most effective at improving accuracy by correcting the errors of previous models and are often preferred to achieve higher accuracy. Overall, the choice between Neural Networks and Decision Trees depend on the specific problem, as well as the resources and time constraints of the task.

3.6 Literature

This summary is mainly based on these four sources, which are not explicitly mentioned in the text:

Beck, Alexander, Blank, Joel, Müntener, Pascal, Pavlics, Arthur, and Kenan Öztürk, 2023, Jupyter Notebooks – ML GROUP PROJECT. (our own Jupyter Notebook)

Zimmermann, Benjamin, 2023, Github Repository *ML_in_Finance* from website, https://github.com/bMzi/ML_in_Finance, 16.04.2023. (documents from seminar lessons)

Zimmermann, Benjamin, 2023, Jupyter Notebooks – Group Project: Minimal Working Expamle. (minimal working example from seminar lessons)

Schmitt, Thomas, 2019, house_prices [data set] from website, https://www.openml.org/search?type=data&sort=runs&id=42165&status=active, 16.04.2023. (housing prices data set)

However, we also considered further literature:

Breiman, Leo, 1984, Classification and regression trees (Routledge, New York).

Breiman, Leo, 2001, Random Forests, *Machine Learning*, Vol. 45 (1), S. 5-29.

Cutler, Richard, Edwards, Thomas C., Beard, Karen H., Cutler, Adele, Hess, Kyle T., Gibson, Jacob, and Joshua J. Lawler, 2007, Random Forests for Classification in Ecology, *Ecology Journal*, Vol. 88 (11), S. 2783-2792.

Chollet, François, 2017, *Deep Learning with Python* (Manning, New York).

Choudhury, Ambika, 2021, AdaBoost Vs Gradient Boosting: A Comparison of Leading Boosting Algorithms from website, https://analyticsindiamag.com/adaboost-vs-gradient-boosting-a-comparison-of-leading-boosting-algorithms/, 16.04.2023.

Mesquita, Deborah, 2021, Python AI: How to Build a Neural Network & Make Predictions from website, https://realpython.com/python-ai-neural-network/, 16.04.2023.

Nielsen, Michael A., 2015, *Neural Networks and Deep Learning* (Determination Press).

Tomek, Ivan, 1976, Two modifications of CNN, *IEEE Transactions on Systems, Man and Cybernetics* 6, S. 769-772.