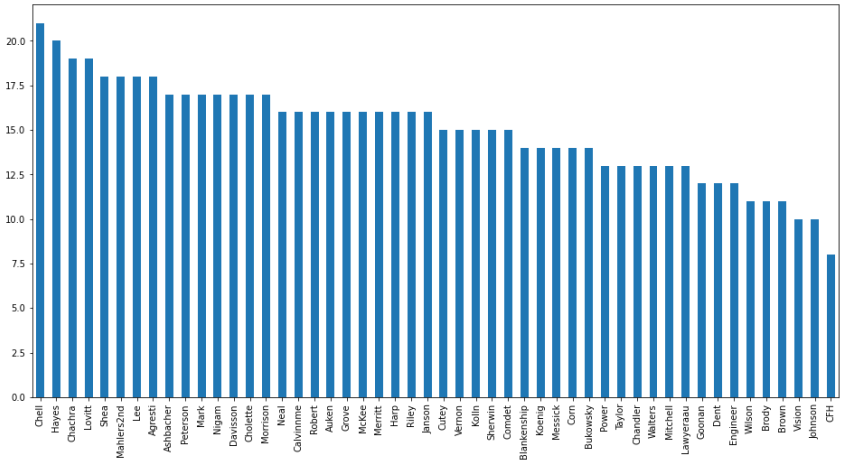
# Exercise 2: Classification

184.702 Machine Learning

Pfeifhofer Lukas (01225541)
Drewitz Olga (01025852)
Merckel Andreas (00746397)

# Dataset: Amazon Reviews

The Amazon Reviews dataset consists of *1500 samples*, split into half into a training dataset and a test dataset. Both datasets contain a *"ID"* column with unique ids from *0 to 1499* and 10.000 columns labeled *"V1" to "V10000"* containing positive integer values. In addition the training dataset has a "Class" column containing 50 unique categories. The goal of the Amazon Reviews dataset is to be able to predict the Author ("Class") for each sample of the test set.




The graph shows the class distribution over the whole training dataset and that it is not balanced. Unbalanced datasets can be a problem and should be dealt with (over-sampling, under-sampling) however the class distribution in the training set might represent the actual class representation in the test dataset so it was decided to keep the samples as they are.

## Preprocessing:

No documentation was to be found on what the columns and their values represent, however they resemble a bags-of-words structure which would mean that each column represents one (or multiple) words of the review content and the number represents how often it appeared in this particular review.
These vectors could directly be used as features for a classifier however it was decided to apply a TF-IDF transformation (term frequency–inverse document frequency) first to better highlight how important a word is compared to all other reviews.
No other preprocessing was applied for the following experiments (no additional form of scaling) as this is known as being not beneficial for features like TF-IDF which by itself already scales the data.
Performance Measures:

- accuracy - This measurement calculates sub-set accuracy (that is the accuracy for each separate class in the multi-class situation)
- balanced_accuracy - This measurement was selected to potentially deal with the imbalance of the dataset (as seen above). It is weighted by average per-class recall.
- f1_weighted - This measurement was also selected because of the potential class imbalance. It calculates the f1 core per-class but weights the result by the amount of existing samples for this class.

## Classifier:

*Random Forest*

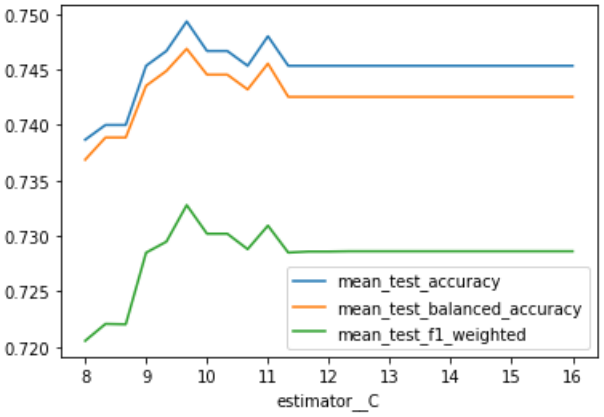
For tuning of the hyper-parameters of the Random Forest classifier a 5-fold CV approach on the training dataset was chosen. For the parameters “n_estimators” (the number of trees/estimators), “max_depth” (the max depth of any tree) and ”criterion” (the function which measures the quality of a split) a range of different values was chosen and then tested. In the table below we can see the best 10 combinations (sorted by the “accuracy”).

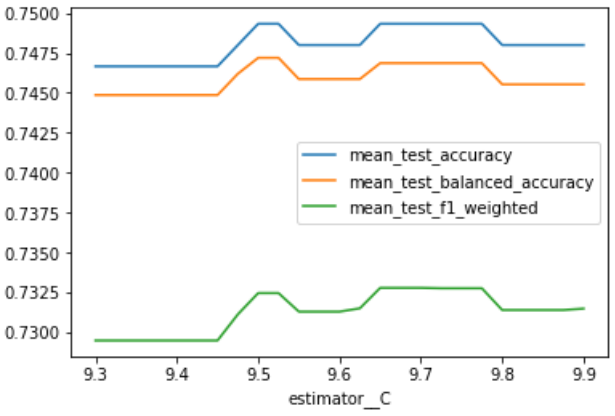
It can also be seen in the table, that if we have a high number of n_estimators (larger number of trees) the time to train the model also scales in the same ratio. Training the final model took ~2.5 minutes.

| | n_estimators | max_depth | criterion | accuracy | balanced_accuracy | f1_weighted | fit_time |
|---|---|---|---|---|---|---|---|
| **19** | 500 | 10 | entropy | 0.657 | 0.654 | 0.628 | 141.933 |
| **24** | 500 | 20 | entropy | 0.652 | 0.652 | 0.631 | 134.144 |
| **29** | 500 | 50 | entropy | 0.635 | 0.633 | 0.613 | 103.837 |
| **18** | 200 | 10 | entropy | 0.611 | 0.614 | 0.584 | 56.721 |
| **23** | 200 | 20 | entropy | 0.608 | 0.608 | 0.578 | 56.863 |
| **28** | 200 | 50 | entropy | 0.599 | 0.598 | 0.574 | 53.851 |
| **27** | 100 | 50 | entropy | 0.537 | 0.534 | 0.509 | 29.185 |
| **22** | 100 | 20 | entropy | 0.525 | 0.524 | 0.504 | 30.800 |
| **14** | 500 | 50 | gini | 0.524 | 0.517 | 0.495 | 160.848 |
| **9** | 500 | 20 | gini | 0.523 | 0.524 | 0.500 | 154.241 |

*SVC*

To find the best possible parameter multiple runs were done towards smaller and smaller ranges of the C parameter to find the optimum. The plots below show all performance measures VS the corresponding C value. The difference between all 3 performance measures is near constant but their value is not. Training took per average ~61s.




*MLP (with PyTorch)*

The last classifier tested for the Amazon Review dataset was a simple MLP-Network. All 10.000 TF-IDF features are taken as input, one hidden layer with 1/10 size (1000) and an output layer of size of 50, one for each category.

The hidden layer used a *ReLu* activation function and the output layer *Sigmoid*.

The optimizer chosen was *Adam* with a default learning rate of 0.01 and the multiclass capeable *CrossEntropyLoss* loss-function was employed. Training the model took about ~77s.

This setup *"MLP (50 epochs)"* will reach an accuracy of 1.0 (for the training dataset) after ~20 epochs of training on the whole training dataset (with only 750 samples). Such a high accuracy on the training dataset often indicates an overfit of the model and could result in poor predictions for new unseen data (in this case the test dataset).
Because of that another network *"MLP (200 epochs)"* with an additional Dropout-Layer after the hidden layer was added. Dropout layers can help against overfitting networks. In addition the network was trained for 200 epochs. Training the model took about ~348s.

### Baseline and Significance Testings

As a baseline classifier a dummy classifier (strategy=Stratified) was selected. A dummy classifier is the simplest classification method and in this case makes predictions based on a simple rule. strategy=Stratified means that it generates random predictions based on the class distribution in the training set.

All experiments were performed with 5-fold CV, the performance score considered for the significance testing is the *"balanced_accuarcy"* to make sure the unbalanced nature of the dataset is taken into account.
We performed a two-tail t-test (4 degrees of freedom) and we tested if the difference between the two means, "balanced_accuarcy(Stratified)" and "balanced_accuarcy", is statistically significant.
If the 95 % confidence interval includes the null-value ("balanced_accuarcy(Stratified)" - "balanced_accuarcy"), then there is a statistically meaningful difference and we reject h0. Otherwise we keep h0 and conclude that there is no significant difference between the two achieved scores.

| Type | balanced_accuracy(Stratified) | balanced_accuracy | balanced_accuracy_sd | t-value | h0 |
|---|---|---|---|---|---|
| **RandomForest** | 0.022333 | 0.653067 | 0.021146 | 66.694932 | reject |
| **SVC** | 0.022333 | 0.746867 | 0.020697 | 78.275825 | reject |
| **MLP (50 epochs)** | 0.022333 | 0.545200 | 0.090529 | 12.914756 | reject |
| **MLP (200 epochs)** | 0.022333 | 0.573691 | 0.031286 | 39.406971 | reject |

As seen in the table we achieved a significant difference to our baseline classifier with all of our trained classifier models.

### Classifier Combination (Majority Voting)

In trying to reach a top score for kaggle the approach was chosen to combine the prediction probabilities of multiple classifiers to achieve a higher accuracy on the test set then a single classifier alone could.
For this approach all probabilities for a sample are added up and finally the category with the overall highest probability is chosen as candidate for the final submission.

With this approach we were able to reach a score of *0.77866* on the 50% of known test samples on kaggle (Corresponds to rank #4 when this report was written).
https://www.kaggle.com/doctorseus/ml-amazonreview-nn-group-25/ (shared with rmayer)

## Summary

It was found that the best performance was achieved with the SVC classifier and a C-Parameter of *9.7*. To increase the accuracy of the prediction it is important to find the best hyper-parameters for a classifier and for MLP classifiers it is also beneficial to avoid over and under-fitting when predicting for unknown data. Training of the classifier takes between 1 and 6 minutes, SVC and the MLP with a lower epoch number are the shortest to train, the MLP with 200 epochs takes the longest.
For the kaggle submission it was shown that the combination of multiple classifiers allowed us to achieve a higher score than with only one of the tested classifiers alone.

# Dataset: SBALoan Approval

The “SBA national” Dataset comes with an accompanying paper to explain the large underlying dataset.

The dataset includes historical data from 1987 through 2014 (899,164 observations) and has 27 features.

## Preprocessing

### Converting values

We converted the dates to a datetime type, the years to integers, the currencies to floats, ‘MIS_Status’, ‘LowDoc’, ‘RevLineCr’, ‘NewExist’, ‘FranchiseCode’, ‘CreateJob’, ‘RetainedJob’, ‘RealEstate’ to categorical. Some of these decisions are guided by the “SBApaper”'. We created ‘recession’ to properly reflect recession times. We created columns for the default rate per state and for SBA approved loans

### Dealing with empty values

We made an attempt to replace these with assumptions based on ‘is_franchised’, but found out that our assumption (“new businesses usually do not have a franchise”) was wrong. So we dropped the missing values in ‘NewExist’. We converted the missing values in ‘LowDoc’ based on the paper-guidelines (If loan < 150k then 0 otherwise 1). For the missing values in ‘MIS_Status’ we used dates in ‘ChgOffDate’

### Dropping categories

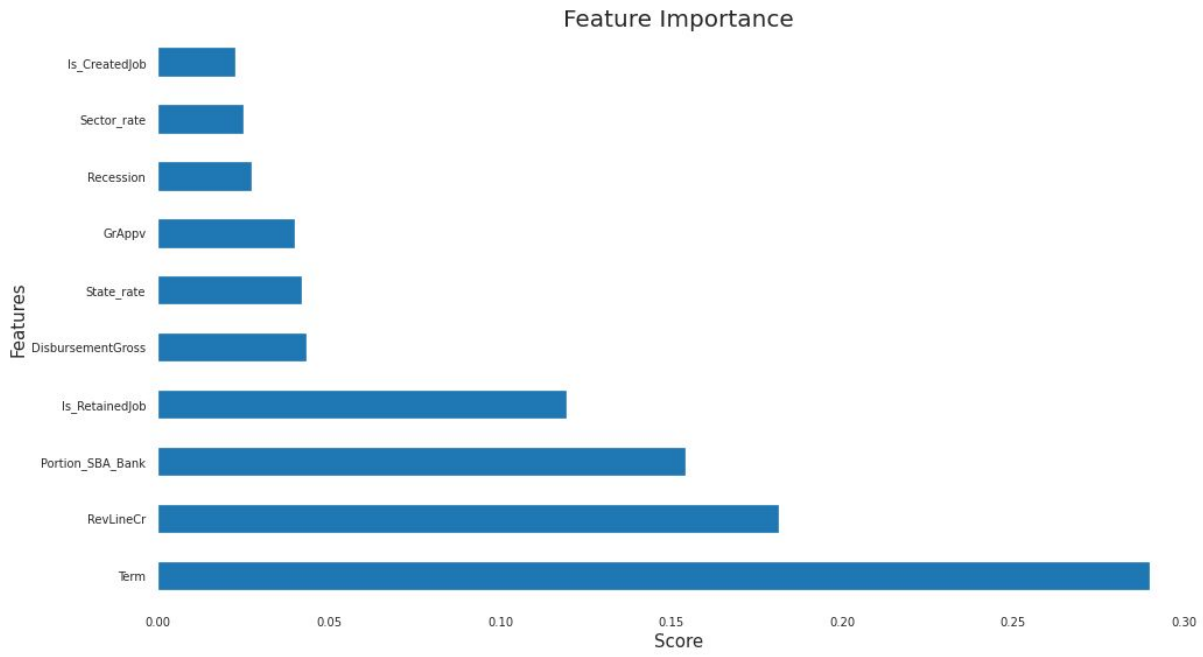
We dropped
LoanNr_ChkDgt: The id of the borrower has been replaced by the index
City, State, UrbanRural, ZIP: We changed to state rates with our own column
bank: The name of the bank is not important for our target
NAICS: replaced with Sector_rate
ApprovalDate, ApprovalFY: These are only internal recording dates
Term: replaced by RealEstate
UrbanRural: does not affect target
LowDoc: irrelevant because of Gross Disbursement
Active, DaysTerm: Replaced by our new recession category
ind_code: equal to Sector_rate
ChgOffDate: equivalent to MIS_Status
DisbursementDate: Only payment dependent
SBA_Appv: Replaced by Portion_SBA_Bank
DisbursementDate: No longer used
Sector_name: no longer used

## Removing outliers

The data has quite a few outliers. We used log transformations in ‘DisbursementGross’, ‘GrAppv’ and dropped the remaining ~1%. We used a boxcox transformation on ‘NoEmp’ and dropped the remaining ~0.02%. We removed the outliers in ‘Term’ that exceeded the underlying timespan from 1987-2010

## Feature selection

Since we have a large number of inputs, we used the feature selection from XGBoost to visualize the importance.



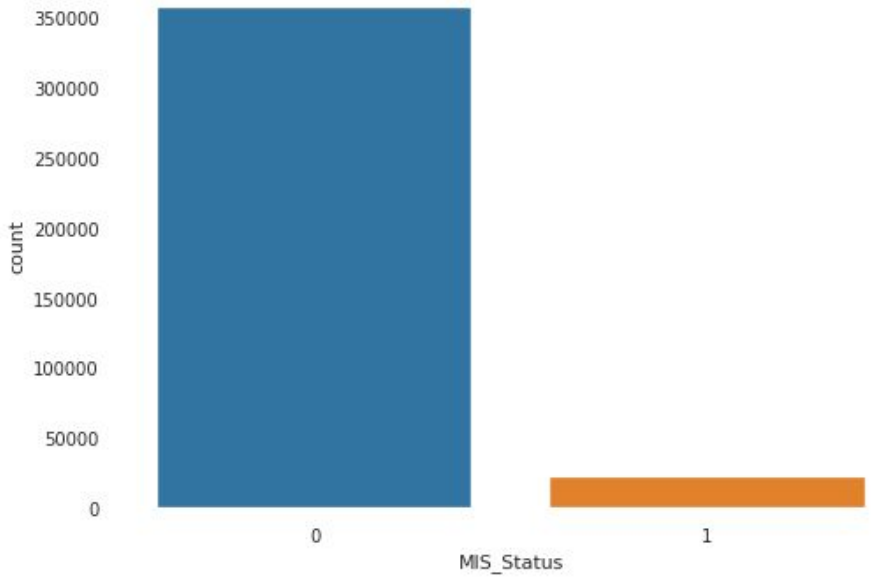

### Theoretical verification of feature selection

1. Term: If a company borrows for a long period of time, the possibility of returning the loan will be greater because the interest will be smaller, making it easier for the company to pay it.
2. RevLineCr:  Revolving Line of credit, i.e. borrowing again and again. If a company is coming back, there is a high possibility for it being a successful company.
3. Portion_SBA_Bank: This is how much the SBA is provided guarantees from the loan provided by a Bank. The greater the percentages, the more the SBA is confident enough with the company not to default.
4. Is_RetainedJob: Does the company have permanent employees? If not, chances are that the company has not been stable because. This increases the possibility of default.
5. DisbursementGross: Important since the amount of the loan is paid whether there is a penalty or according to the initial loan, if this is more it is likely that there is a penalty.
6. State_rate: Each area's policy is different and this is calculated by the means of each region over the defaults from all sectors, so the higher the more likely it is to default.

7. GrAppv = Represents the number of loans given by banks. Since banks do their own research, we consider this important outside information.
8. Recession = If the company was able to stay active when times were hard like in a recession, then there is a higher possibility of the business to be stable.
9. Sector_rate = Like the State rate, each sector also has its own default rate or percentage of defaults.
10. Is_CreatedJob = By offering employment it can be concluded that a company is developing and if a company develops it will reduce the possibility of default.

## Unbalanced Dataset

As we can see, our target variable is highly unbalanced, with 0 (not default): 94.3% and 1 (default): 5.7%. In order to deal with this imbalance, we divided the dataset in two




## Classifier:

To find out the importance of the features for the target variable, we used a variety of classifiers, first in a non-optimized version in order to select which classifier would be the most useful for further exploration, then we selected the best one to optimize.

### Logistic Regression:

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.975 | 0.597 | 0.740 |
| 1 | 0.097 | 0.736 | 0.172 |
| accuracy | | | 0.604 |
| macro avg | 0.536 | 0.666 | 0.456 |
| weighted avg | 0.926 | 0.604 | 0.708 |

### Naive Bayes

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.983 | 0.529 | 0.688 |
| 1 | 0.096 | 0.849 | 0.173 |
| accuracy | | | 0.547 |
| macro avg | 0.540 | 0.689 | 0.431 |
| weighted avg | 0.934 | 0.547 | 0.659 |

### KNN

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.978 | 0.916 | 0.946 |
| 1 | 0.316 | 0.654 | 0.426 |
| accuracy | | | 0.902 |
| macro avg | 0.647 | 0.785 | 0.686 |
| weighted avg | 0.941 | 0.902 | 0.917 |

### Random Forest

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.982 | 0.967 | 0.974 |
| 1 | 0.554 | 0.694 | 0.424 |
| accuracy | | | 0.952 |
| macro avg | 0.768 | 0.831 | 0.795 |
| weighted avg | 0.958 | 0.952 | 0.954 |

### XGBoost

This classifier uses gradient boosting to produce an ensemble of weak predictions models, i.e. decision trees.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.987 | 0.965 | 0.976 |
| 1 | 0.571 | 0.782 | 0.660 |
| accuracy | | | 0.955 |
| macro avg | 0.779 | 0.874 | 0.818 |
| weighted avg | 0.964 | 0.955 | 0.958 |

As we can conclude from these tables, the ‘XGBoost’ classifier is the most useful.

### Optimization of Classifier

In order to optimize our classifier we used the f-score and GridSearchCV. This gave us optimized parameters for XGBoost. We plucked the optimal hyperparameters into another run and got the following:

### XGBoost optimized

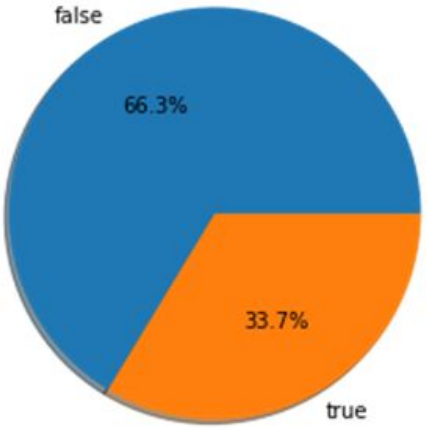
| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.990 | 0.951 | 0.970 |
| 1 | 0.501 | 0.838 | 0.627 |
| accuracy | | | 0.944 |
| macro avg | 0.746 | 0.894 | 0.799 |
| weighted avg | 0.963 | 0.944 | 0.951 |

## Summary

It was found that the best performance was achieved with the XGBoost classifier, both optimized and non-optimized. The worst performing classifier was the Naive Bayes. The dataset needs extensive pre-processing and quite a lot of economic understanding and assumptions. It is highly likely that having a deeper understanding of the economic principles behind the SBA-processes and american small business in general can lead to better predictions and outcomes.

# Dataset: Breast Cancer

The dataset consists of 569 samples, split into a training set (285 samples) and a test set (284 samples). There are 32 columns in the training set and 31 columns in the test set (the test set is without the target feature). The goal is to predict the Breast Cancer feature 'MALIGNANT' which is either True or False.




As one can see in the plot, the distribution is not balanced. There are more samples where 'MALIGNANT' = false.

## Preprocessing

The dataset has no missing values. There was only one column named 'ID' removed, beauce it has only unique values. No further preprocessing was done.

## Performance Measures

- accuracy: This measurement calculates the accuracy classification score.
- precision: The precision is the ability of the classifier not to label as positive a sample that is negative
- recall: The recall is the ability to find all the positive samples.

## Classifier

### Random Forest

For tuning of the hyper-parameters and Scalers of the Random Forest classifier a 3-fold CV approach on the training dataset was chosen. For the following parameters a range of different values was chosen and then tested:

- n_estimators (the number of trees/estimators): 10, 50, 100, 200
- criterion (the function which measures the quality of a split): gini, entropy
- max_depth (the max depth of any tree): None, 20, 30, 50

The best result was: n_estimators = 50, criterion = 'gini', max_depth = None

### SVC

The SVC already showed good performance (on accuracy) with the default parameters. For tuning of the parameters a similar approach as for the Random Forest classifier was chosen (3-fold CV).
There were several Parameter grids tested:

| | C | Gamma | kernel | best choice |
|---|---|---|---|---|
| 1. | [x for x in np.linspace(start = 0.1, stop = 1.0, num = 25)] | auto, scale | linear, poly sgmoid, rbf | C = 0.1<br>gamma = auto<br>kernel = poly |
| 2. | [x for x in np.linspace(start = 0.1, stop = 1.0, num = 10)] | auto | poly | C = 0.1 |
| 3. | [x for x in np.linspace(start = 0.01, stop = 0.11, num = 10)] | auto | poly | C = 0.01 |

Additionally different Scalers were tested: No Scaler, Standard Scaler and Robust Scaler.

| | No Scaler | Standard Scaler | Robust Scaler |
|---|---|---|---|
| accuracy | 0.9614 | 0.7263 | 0.7193 |

### NuSVC

The difference to the SVC Model ist, that it uses a parameter to control the number of support vectors.The NuSVC already showed good performance (on accuracy) with the default parameters. For tuning of the parameters a similar approach as for the other classifier was chosen (3-fold CV).

- nu: [x for x in np.linspace(start = 0.01, stop = 0.99, num = 100)]

Best choice: 'nu = 0.06939393939393938
Additionally different Scalers were tested: No Scaler, Standard Scaler and Robust Scaler.

| | No Scaler | Standard Scaler | Robust Scaler |
|---|---|---|---|
| accuracy | 0.95088 | 0.9754 | 0.9754 |

### MLP

For tuning of the parameters of the MLP a 3-fold CV approach on the training dataset was chosen.

- max_iter: 10, 50,100,150, 200, 300,1000
- hidden_layer_size: (2,2),(3,3), (10,4)

The best choice was: max_iter = 1000, hidden_layer_sizes: (3,3)

## Baseline and Significance Testings

As a baseline classifier two different choices were choose:

· All false: the majority class is predicted for every sample

· Random classifier: the class is predicted by chance

| Type | accuracy | precision | recall |
|---|---|---|---|
| **Baseline All false** | 0.663158 | 0.000000 | 0.000000 |
| **Baseline All random** | 0.578947 | 0.403226 | 0.520833 |
| **RandomForest** | 0.954386 | 0.948440 | 0.916667 |
| **SVC** | 0.961404 | 0.958586 | 0.927083 |
| **NuSVC** | 0.975439 | 0.978472 | 0.947917 |
| **MLP** | 0.635088 | 0.395109 | 0.625000 |

| | Type | false | false % | true | true % |
|---|---|---|---|---|---|
| **0** | Baseline All false | 285.0 | 100.000000 | 0.0 | 0.000000 |
| **1** | Baseline All random | 161.0 | 0.564912 | 124.0 | 0.435088 |
| **2** | RandomForest | 173.0 | 61.000000 | 111.0 | 39.000000 |
| **3** | SVC | 180.0 | 63.000000 | 104.0 | 37.000000 |
| **4** | NuSVC | 174.0 | 61.000000 | 110.0 | 39.000000 |
| **5** | MLP | 282.0 | 99.000000 | 2.0 | 1.000000 |

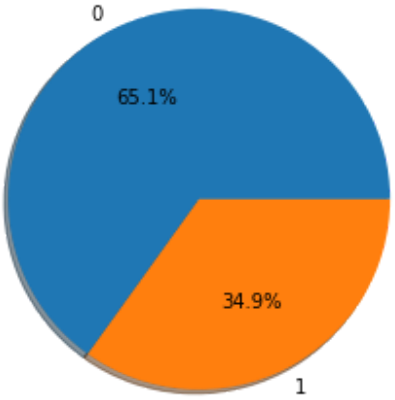
## Summary

The best performing model was the NuSVC model with an accuracy of 0.975. The lowest accuracy was achieved in the MLP model with 0.635, which is even lower than the Baseline with “ALL false” answers. The dataset was small and hence the computation time was low. The most time consuming was the grid testing for the SVC Model with different kernels. It is also interesting that during the testing periode the best model and the best parameter choice changed nearly every run.

# Dataset: Diabetes

The dataset consists of 768 samples and 9 columns. The goal is to predict whether or not someone has diabetes. There are 8 medical features and the target feature outcome.

As one can see in the plot, the distribution is not balanced. There are more samples without diabetes (outcome = 0) than with diabetes (outcome = 1).




## Preprocessing

There was no preprocessing done. The data has no missing values and since there was a 3-fold Cross Validation used, no splitting in test and train data was needed.

## Performance Measures

- accuracy: This measurement calculates the accuracy classification score.
- precision: The precision is the ability of the classifier not to label as positive a sample that is negative
- recall: The recall is the ability to find all the positive samples.

## Classifier

### Random Forest

For tuning of the hyper-parameters and Scalers of the Random Forest classifier a 3-fold CV approach on the training dataset was chosen. For the parameters “n_estimators” (the number of trees/estimators), “max_depth” (the max depth of any tree) and ”criterion” (the function which measures the quality of a split) a range of different values was chosen and then tested:

- n_estimators: 10, 50, 100, 200
- criterion: gini, entropy
- max_depth: None, 20, 30, 50

The best result was: n_estimators = 50, criterion = ‘entropy’, max_depth = 20

### SVC

The SVC already showed good performance (on accuracy) with the default parameters. For tuning of the parameters a similar approach as for the Random Forest classifier was chosen (3-fold CV).
Different values for C were tested:

- C: [x for x in np.linspace(start = 0.01, stop = 0.99, num = 100)]

The best result was for C=0.9108080808080807

Additionally different Scalers were tested: No Scaler, Standard Scaler and Robust Scaler.

| | No Scaler | Standard Scaler | Robust Scaler |
|---|---|---|---|
| accuracy | 0.7513 | 0.7734 | 0.7723 |

### NuSVC

The difference to the SVC Model ist, that ituses a parameter to control the number of support vectors.The NuSVC already showed good performance (on accuracy) with the default parameters. For tuning of the parameters a similar approach as for the other classifier was chosen (3-fold CV).

- nu: [x for x in np.linspace(start = 0.01, stop = 0.99, num = 100)]

Best choice: 'nu = 0.5643434343434344

Additionally different Scalers were tested: No Scaler, Standard Scaler and Robust Scaler.

| | No Scaler | Standard Scaler | Robust Scaler |
|---|---|---|---|
| accuracy | 0.7526 | 0.7682 | 0.7695 |

### MLP

For tuning of the parameters of the MLP a 3-fold CV approach on the training dataset was chosen.

- max_iter: 10,50,100,150,200,300, 1000
- hidden_layer_size: (2,2),(3,3), (10,4)

The best choice was: max_iter = 1000, hidden_layer_sizes: (3,3)

## Baseline and Significance Testings

As a baseline classifier two different choices were choose:

- All false: the majority class is predicted for every sample
- Random classifier: the class is predicted by chance

| Type | accuracy | precision | recall |
|---|---|---|---|
| **Baseline All false** | 0.651042 | 0.000000 | 0.000000 |
| **Baseline All random** | 0.488281 | 0.323944 | 0.429104 |
| **RandomForest** | 0.768229 | 0.719230 | 0.566916 |
| **SVC** | 0.766927 | 0.726188 | 0.544694 |
| **NuSVC** | 0.759115 | 0.715532 | 0.522222 |
| **MLP** | 0.651042 | 0.166667 | 0.003704 |

| | Type | false | false % | true | true % |
|---|---|---|---|---|---|
| **0** | Baseline All false | 768.0 | 100.00000 | 0.0 | 0.00000 |
| **1** | Baseline All random | 413.0 | 0.53776 | 355.0 | 0.46224 |
| **2** | RandomForest | 500.0 | 65.00000 | 268.0 | 35.00000 |
| **3** | SVC | 562.0 | 73.00000 | 206.0 | 27.00000 |
| **4** | NuSVC | 565.0 | 74.00000 | 203.0 | 26.00000 |
| **5** | MLP | 767.0 | 100.00000 | 1.0 | 0.00000 |

## Summary

The best performing model was the random forest model. Although the accuracy is quite low with 0.768. The worst Model was the MLP model with an accuracy of 0.651 and a very low precision and recall score. Since the dataset is small, the evaluation and prediction of the dataset was fast (under 5s computational time). Also it is interesting that scaling has nearly no effect on the model outcome.