Report Exercise 1: Classification

Jürgen Brandl (01527070)
Tobias Hajszan (11776172)
Moritz Staudinger (11777768)

Introduction

For this exercise, we evaluate the performance of four different classification machine learning algorithms based on different datasets. We will therefore use the algorithms k-nearest neighbors, random forest, support vector machine and stochastic gradient descent. These algorithms will be used in different variations to classify the following datasets:

- Phishing Websites (11055 rows, 31 features)
  https://www.openml.org/d/4534

- Zoo (100 rows, 17 features)
  https://www.openml.org/d/62

- Amazon Reviews (1500 rows, 10001 features)
  https://www.kaggle.com/c/184702-tu-ml-ws-20-amazon-commerce-reviewss

- Congressional Voting (435 rows, 17 features)
  https://inclass.kaggle.com/c/184702-tu-ml-ss-20-congressional-voting

With the differences in the number of rows and features, as well as the encoding and datatype of each of these datasets, we hope to get valuable insights in how these algorithms work and how much information they need to perform consistently.

Our work is structured as follows, at the start we will explain how these algorithms work, and then we will describe each dataset and evaluate the performance of the algorithms on them. In the end, we will sum up our findings and what we have learned with this task.

Algorithms

We have chosen the following algorithms to be able to make comparisons across all datasets. Therefore we use different variations of each algorithm, but for comparison reasons, we use the same parameters for each dataset.

k-nearest neighbors

About

The k-nearest neighbor algorithm, is a learning algorithm which classifies nodes/individuals based on their neighborship. Each neighbor owns a voting right to vote for the class of the individual, which needs to be classified.Therefore the distance between each node and the node to be classified is determined and then the nearest k-nodes are voting on its classification.

It is necessary to choose a good value k for the amount of neighbors, so that our error-rate is as small as possible, but we also do not want our model to over- or underfit the model.

For the weight each node has in the voting there are two different approaches. On the one hand there is the uniform weighting, where each node has the same influence on the result and on the other hand there is the distance weighting, where the nodes impact the result based on the distance to the node.

Why this algorithm?

The KNN algorithm is an instance based learning algorithm (Lazy Learner). This means that this algorithm has no training period and learns only from it, when he is making predictions. Therefore this algorithm is really fast and new data can be added easily.
(http://theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of-knn.html , https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering)

Evaluation

For this algorithm the accuracy for a combination of weight, distance metric [uniform, euclidean], algorithm [ball-tree, kd-tree, brute-force], number of neighbours [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 50, 100, 200, 500, 1000, 2500] was evaluated.

Random Forest

About

The random forest algorithm is an ensemble tree-based learning algorithm, which consists of a set of decision trees from subsets, which are retrieved randomly. Therefore it samples subsets from the testset and builds a decision tree on each of these subsets. Afterwards all of these subsets are gathered together and base a decision for the prediction on a majority vote for each subset.
The precision of this algorithm is based on the number of decision trees, which are generated and the depth the decision tree can have. For our example we have used a max depth of 1 because the calculation time increased heavily with the larger datasets. Furthermore the performance between the different runs vary, as the subsets are sampled randomly and therefore the decision trees are different.

Why this algorithm?

The Random Forest algorithm is one of the most accurate learning algorithms and classifies therefore really well and should improve. It can handle large amounts of input variables efficiently and can also estimate missing values. It may overfit data, if there is too much noise in the data, which it tries to circumvent by aggregating the results of the different trees. (https://medium.com/swlh/random-forest-classification-and-its-implementation-d5d840dbead0)

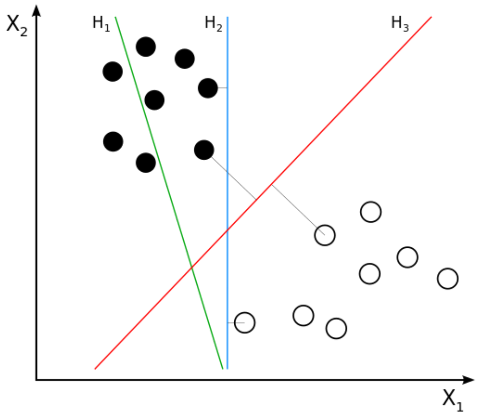
Evaluation

For this algorithm the accuracy, precision, recall and fbeta for the number of trees [1,2,3,4,5,6,7,8,9,10,15,20,25,30,50,100,200,500,1000,2500] was evaluated.

Support Vector Machine

About

The support vector machine is a type of learning classifier algorithm, which tries to draw decision boundaries between data points to classify these into classes. With each boundary drawn, the data points are divided in more parts, which are from a different class. The goal when placing each hyperplane is to maximize the distance between the two classes, so that the values are predicted as good as possible.



These hyperplanes can be linear, polynomial or on the basis of a radial function and also can take a value for its penalty function in the range [0.1,100.000] into account.

Why this algorithm?

This algorithm works really well when there is a clear margin of separation (eg. 1 and 0) and is also effective in high dimensional spaces. On the other hand there are problems with the performance on larger datasets as more training time is needed and when the classes are overlapping.
(https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/)

Evaluation

For this algorithm the accuracy for a combination of kernel [poly, rbf, linear], penalty [0.1, 1, 10, 100, 1000], gamma [auto, scale] and the degree [0,1,2] was evaluated.

Stochastic Gradient Descent

About

Stochastic Gradient Descent is a method, which is a special form of the Gradient Descent algorithm. The Gradient Descent method is an optimization technique, which uses the degree of change of a variable in response to the changes of another variable.
This gradient is simply speaking the slope of a function, the steeper the slope the greater the gradient.
From an initial start value, this method runs iteratively to find the minimum possible value for a given cost function.

The Stochastic Gradient Descent brings randomness to this approach, as it is not feasible for larger datasets to calculate all values for this algorithm as this would be too expensive. Therefore only a few randomly selected samples are selected instead of the whole set for each iteration.

Why this algorithm?

We have chosen this algorithm, as it is not as costly to perform as the original Gradient Descent algorithm and is better fitted for larger datasets. As we have not picked a larger dataset its full potential can hardly be shown in our classification tasks.

Evaluation

For this algorithm the accuracy for a combination of loss [hinge, squared_loss, huber, epsilon_insensitive, squared_epsilon_insensitive], penalty [l1, l2, elasticnet], max iterations [10^(6, 6.5, 7) / length of data] and alpha [0.0001, 0.001, 0.01, 0.1] was evaluated.

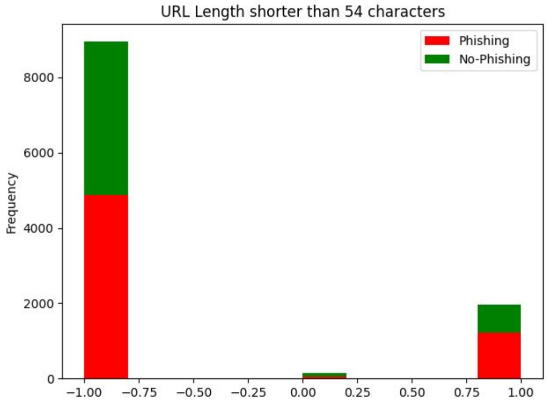
Phishing Websites

Description

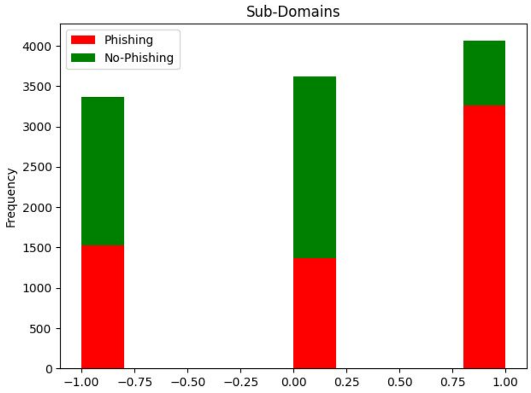
This dataset is a representation of websites both phishing and non-malicious. It's based on a research paper, which presents most publicly available datasets on phishing websites as unreliable and introduces a method that based on the normalization of different features tries to introduce a novel approach to extract more dependable learning data from those websites.

This set provides 11.055 rows of samples with 31 different features which some are described in more detail below. To make analysis easier, there has already been extensive preprocessing done and reduced to binary values. So we are already dealing with values being either -1, 0 or 1. There are no missing values in this dataset either.

Url length: The average url length in this whole dataset is 54 characters. So the dataset was split into 3 categories: below 54, between 54 and 75 as well as above 75. As we can see, there is no clear correlation, but long urls do more often correlate to phishing websites.

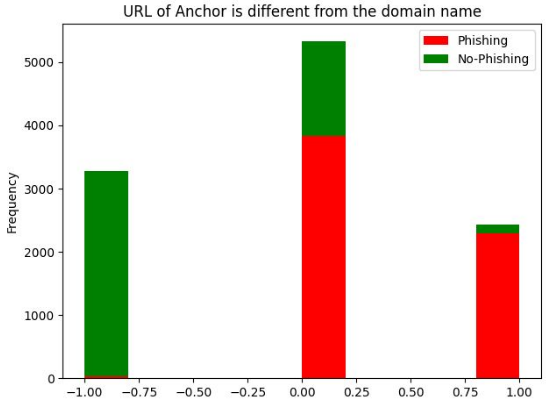


Sub Domains: Depending on the number of subdomains, so none, one or more than one, the feature subdomains becomes either -1, 0 or 1. As we can see there is no clear distinction for none or one, but a slight increase in more subdomains associated with phishing.

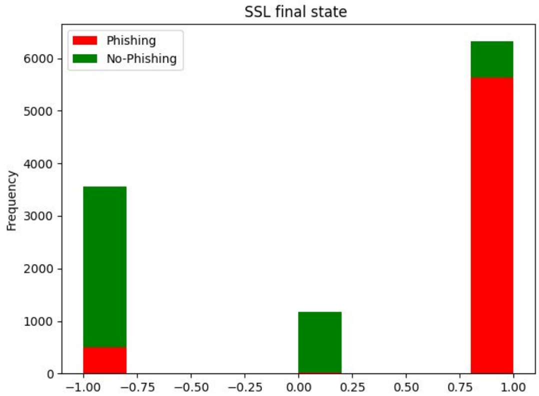


Now for some examples of features, where the is a stronger more obvious correlation to phishing:

Url of anchor is different than domain name: So if links are pointing not towards an url but are rather Javascript, measured in thirds of all urls on the page:

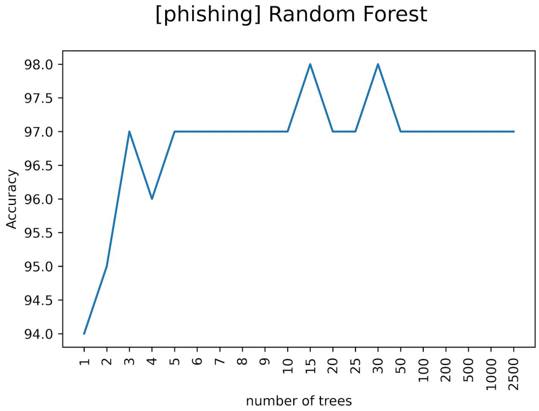


SSL final state: SSL Certificates are deemed more trustworthy, if the issuer is well known and the certificate older than one year. Interestingly enough, this provides a very good prediction on phishing or nonmalicious:



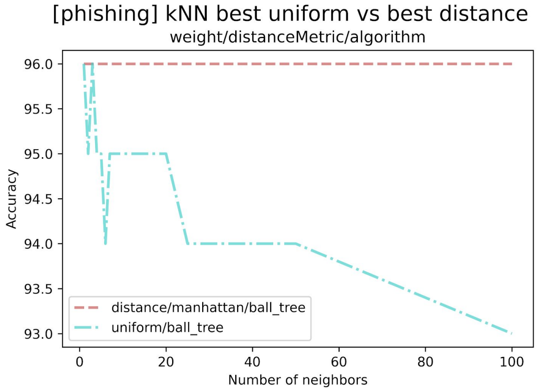
Classification Results and Evaluation

Random Forest



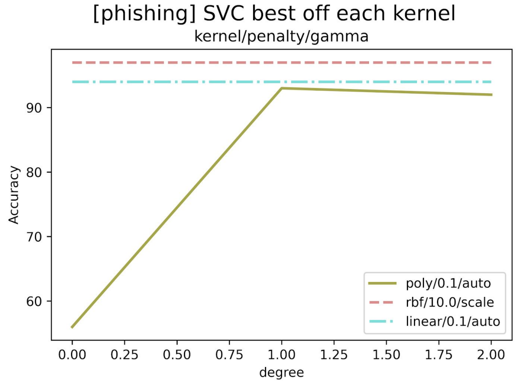
The random forest approach works well on the phishing dataset. I can predict the test data up to 99% correct. With one tree, the result is only 94% correct, but with every tree added it can improve its predictions up to 97% accuracy with 3 trees. At 5 trees up to 2500 trees, the results stay flat with the exception of two very good results of 99% which could be explained by the randomness of this approach. Overall the results are very good, more than 100 trees are probably not really necessary and not worth the calculation overhead.

K-Nearest Neighbours



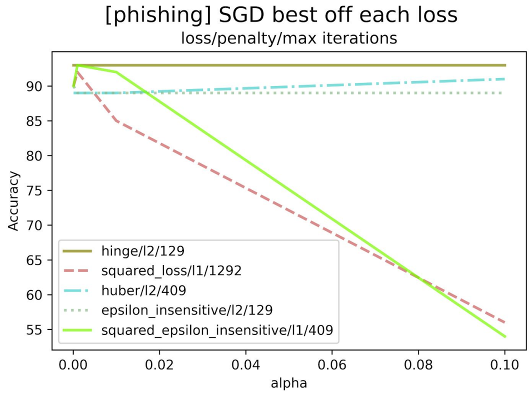
The k-nearest neighbours' approach works also pretty well on this dataset. We analyzed different combinations of distance metrics and algorithms and then took the best of each weight. For the "distance" weight the best combination was to use the manhattan distance metric combined with the ball_tree algorithm.The accuracy stays at 96%. With "uniform" weight, the results were overall much better. This approach could achieve 96 % accuracy with 1 neighbor. The more neighbors were added, the worse the result got.

Support Vector Machine



The support vector machine approach also delivers good results. Here we analyzed combinations of kernel, penalty, and gamma and plotted the accuracy against the degree of the function. For every kernel, the best combination is plotted against each other on the left. The linear and rbf approach are both producing good accuracy with 97% and 94%. The polynomial approach delivers, unsurprisingly, worse results with a degree of 0, but can also deliver good results with 1 which results in a linear like approach. With 2 degrees the accuracy is slightly worse than the other two kernels. All runs performed best with 0.1 penalties and gamma set to "auto", just rbf performed best with "scale".

Stochastic Gradient Descent



The stochastic gradient descent can also produce good results with the right parameters. We evaluated a combination of loss, penalty, and max iterations and plotted the accuracy against the alpha value of the function. With an alpha value of 0.0001 all loss combinations could produce over 88% accuracy. The bigger the alpha gets the worse the results for the loss function of squared loss and squared epsilon insensitive, which all drop under 60% accuracy with an alpha of 0.1. Only hinge, huber and epsilon_insnsitive stayed the same only huber increasing a little over the increase of alpha.
hinge and epsilon insensitive worked best with 129 maximum iterations and l2 penalty, whereas squared epsilon insensitive and huber got better results with 409 iterations. squared loss is the only algorithm that performed better with over 1000 max iterations and l1 penalty.

Zoo

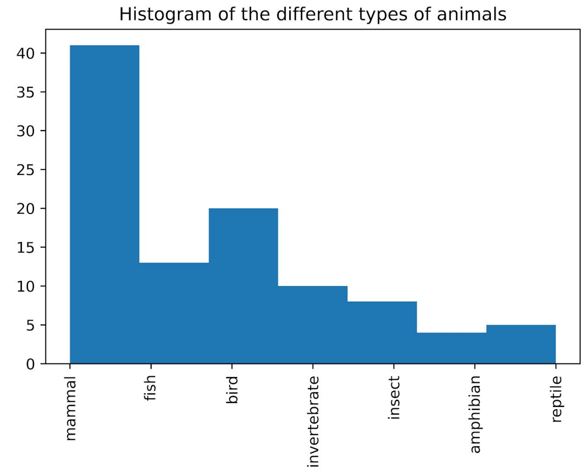
https://www.openml.org/d/62

Description

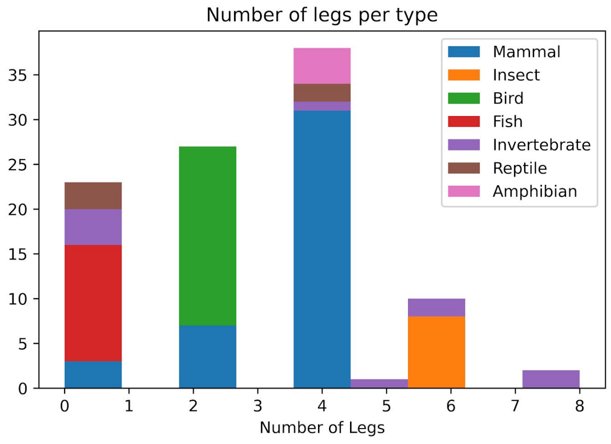
This dataset represents a collection of animals, which are categorized after 17 different characteristics. Most of these categorizes are boolean values, as for example if the animals have hairs or are predators. There are two more advanced categories with the number of legs and the type of animal, which are described below in more detail.

This set provides 101 rows of samples with 17 different features which some are described in more detail below. The data has not been preprocessed by the provider, but we have mapped all True/False values to 1 and 0. So we are already dealing with values being either numbers or are preprocessed to 1 and 0. There are no missing values in this dataset either. As each animal only appears once in the dataset it is not possible to predict the animal based on characteristics, but only the type of the animal. For a classification of the animals, it would be necessary to have characteristics, which differ between the different individual animals, such as the height, length, age, country they are living in and so on.

We can see that 40% of the animals in this dataset are mammals, whereas only a small fraction of 4% are amphibian. With this distribution of the values, which we are going to predict based on our characteristics, being so uneven, this can lead to problems in predicting new animals of the types with smaller samplesizes. this does not seem to be a completely unbiased sample, as this can lead to wrong predictions.



The other interesting characteristic to talk about, is the amount of legs, each animal has. In the plot we can see that each type of animal has a special amount of legs. With this in mind, if we perform a classification based on the legs, which gets us valuable information about the distribution. So we can see that nearly all of the animals with 4 legs are mammals and all birds have exactly 2 legs.

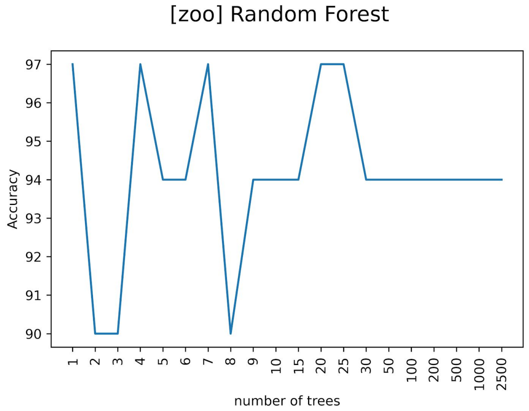


An interesting feature here is that the invertebrate species has several different amounts of legs, and is special, because it has one entry with five legs. We were thinking of misclassification at first, but after a second look and a bit of research, we found out that this is normal for a starfish.

Classification Results and Evaluation

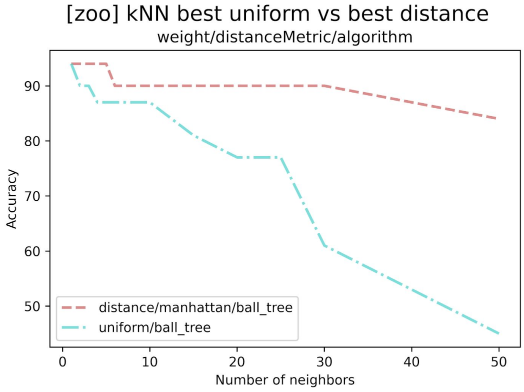
Random Forest

The random forest approach can deliver good results with the zoo dataset, but they tend to vary much with a tree count under 10. Starting with 1 tree, the algorithm could score 97% accuracy, but with 2 and 3 trees only 90% of the prediction was right. 4 trees again deliver a very good result. These variations with so little trees are probably so different because of the randomness of this approach, at 10 trees the predictions become more robust and dont fall above 94% accuracy.



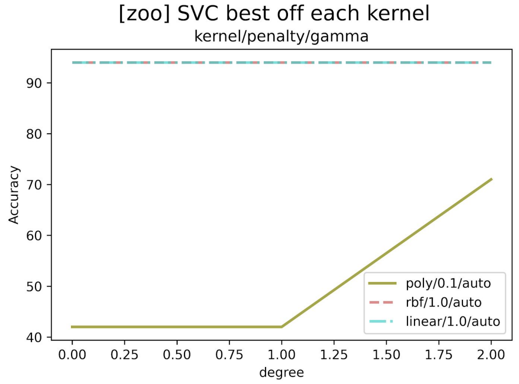
K-Nearest Neighbours

The k-nearest neighbours approach also gives pretty good predictions. We again analyzed combinations of distance metrics and algorithms and took the best resulting combination of each weight. With “uniform” weight we could achieve over 96 % accuracy with just 1 neighbour. The more neighbours where added, the worse the result got for both uniform and distance.
For the “distance” weight the best distance Metric was euclidean distance metric combined with the ball_tree algorithm. the results were overall better, but it can't beat the best result of distance.



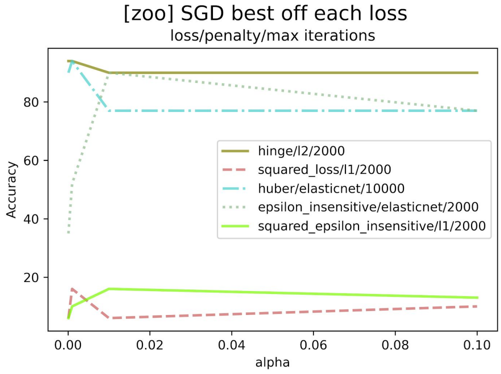
Support Vector Machine

The support vector machine approach delivers both good and bad results. Again, combinations of kernel, penalty and gamma are shown . The accuracy is plotted against the degree of the function. The poly approach is producing bad accuracy of about 42%. The linear and rbf approach deliver the best results with about 94% accuracy. With 2 degrees the accuracy of the polynomial approach rockets to about 70%. Rbf and linear performed best with 1.0 penalty and gamma set to “auto”, poly worked better with 0.1 penalty. Here a further evaluation of higher degrees for the polynomial approach would be interesting.



Stochastic Gradient Descent

The stochastic gradient performs best with the right parameters. Again a combination of loss, penalty and max iterations was used. The accuracy was plotted against the alpha value of the function. The combinations behave very differently on the variation of alpha. The all time high of over 97% could be achieved with hinge loss, l2 penalty and 2000 max iterations, the bigger alpha the worse the result until it flattened out at about 90%. Huber could also reach the all time high, but otherwise the results are rather bad compared to hinge. epsilon_insensitive is pretty good as soon as alpha is over 0.01, but does not get any right results with alpha of 0.0001. Both squared loss functions perform poorly.



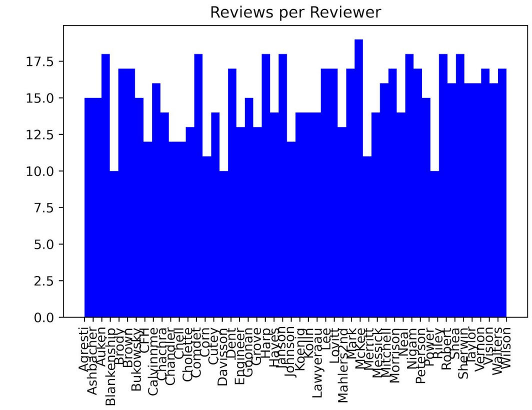
Amazon Reviews

Description

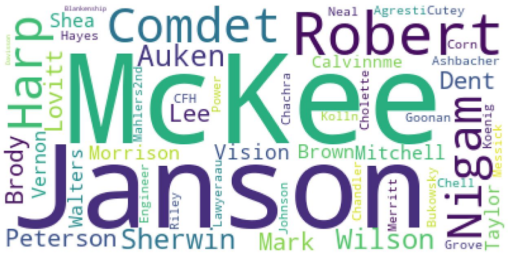
This dataset represents a collection of amazon reviews, which have been made by different users and are categorized after 10.000 different characteristics. All of these categories are numeric values, This set provides a total of 1500 rows of samples, which represent 50 different people reviewing.
There is no preprocessing in the dataset needed, as all data is already numeric.

The dataset is hard to analyze, as we can analyze different columns in the dataset, but as they are all named from 1 to 10.000 it is not possible to gain any information out of them. That is why we have only analyzed this dataset based on the amount reviews per reviewer, and represented it in two different plots, as we wanted to experiment with a different representation with the data as well, which is easier understandable.

We can see that in the training set the names McKee, Janson, Robert, Comdet, Nigam and Harp appear the most often, as these have the highest values in the bar plot as well as also the biggest font size in the word cloud.



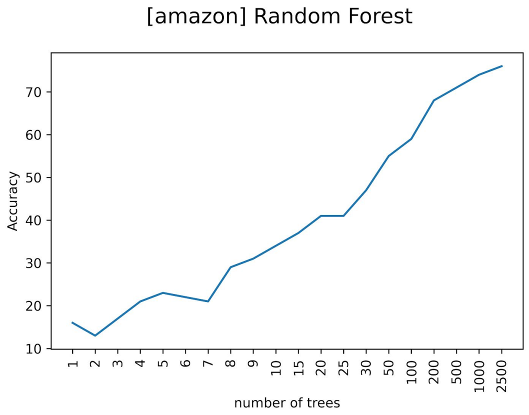
Sadly we need to state that the word cloud was not made for the scientific representation of the data and the sizes are not all according to its true frequency in the dataset, as the values are fitted according to the size in the end and the needed space is not calculated beforehand..



Classification Results and Evaluation

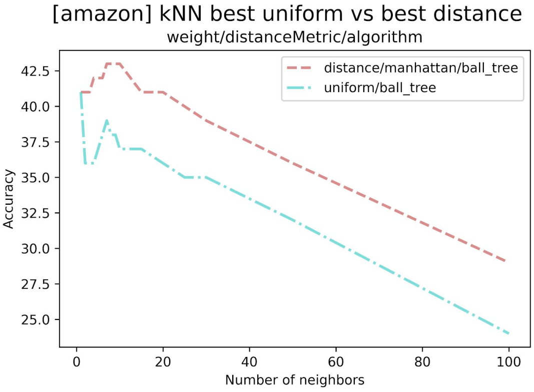
Random Forest

The random forest approach can deliver good results with the amazon dataset. The improvement over the increment of trees cannot be overlooked. Starting low under 20% random forest can improve with increasing trees up to 75% with 2500 trees. The improvement gets more and more shallow. The x-scale here is not linear. The line looks promising for even better results, but our computers are not capable of computing with more than 2500 trees as computer memory blocks and computation time bottlenecks the calculation.



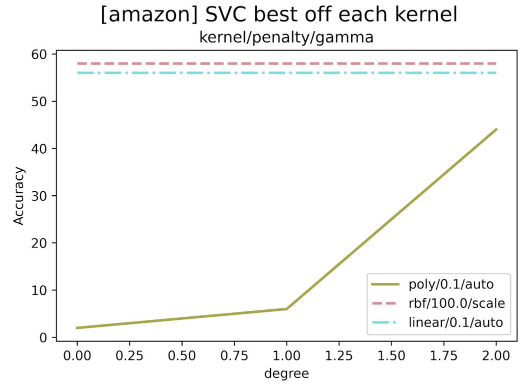
K-Nearest Neighbours

The k-nearest neighbor algorithm performs not great on this dataset in comparison to the other algorithms. We have chosen the best settings for distance and uniform weighting and compare them in the chart on the right. The best performing distance setting gets about 43% of the predictions correct, with the usage of a ball-tree algorithm, manhattan for distance-metric and 15 neighbors. The best uniform setting is performing up to around 41% with 1 neighbor.
After both peaks, we can see that the algorithms are nearly steadily decreasing with the number of neighbors increasing.



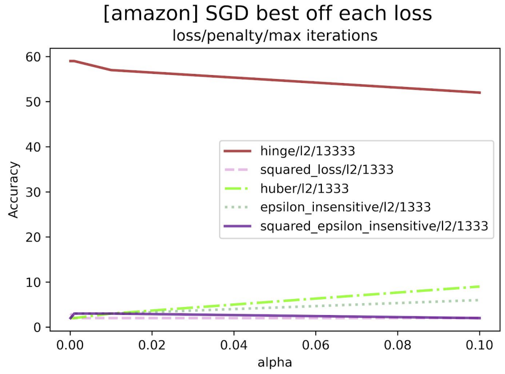
Support Vector Machine

The support vector machine approach delivers both okay and really bad results. Again, combinations of kernel, penalty and gamma are shown . The accuracy is plotted against the degree of the function. The poly approach is producing bad accuracy of about 3%. The linear approach and rbf deliver the best results with about 57% accuracy. With 2 degrees the accuracy of the polynomial approach rocketed. Poly and linear performed best with 0.1 penalty and gamma set to “auto”. Rbf with 100 penalty and gamme set to “scale”. Here a further evaluation of higher degrees for the polynomial approach would be interesting.



Stochastic Gradient Descent

The stochastic gradient performs best with a hinge loss, 13333 max iterations and l2 penalty. The other combinations of loss, penalty and max iterations were far behind. The all time high of just under 60% could be achieved with hinge loss and small alpha, with alphas higher than 0.001 the results worsened. With increasing alpha the accuracy of huber and epsilon insensitive improves slightly. The other two options (squared loss and squared epsilon insensitive) remain nearly on the same level of accuracy. All together hinge loss is the only usable option here, as the other settings do not even surpass 10% accuracy.

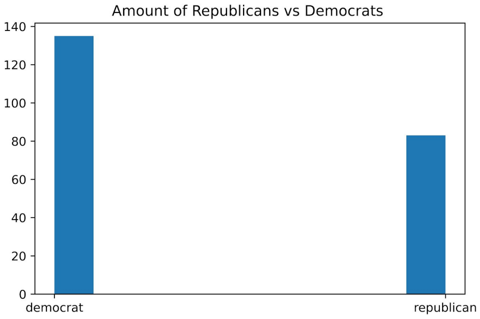


Congressional Voting

Description

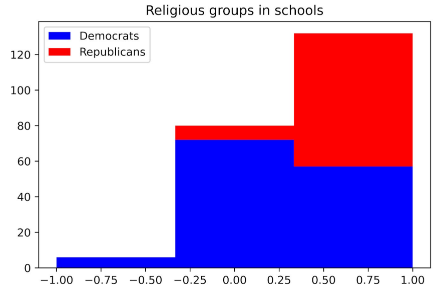
This dataset represents a collection of votes for the US congress, which are categorized after 16 different characteristics. All of these categories are boolean values, This set provides a total of 435 rows of samples, which represent around 265 democrats and 170 republicans (precise numbers are unknown) over the complete set and 130 to 80 in the learning set.

For each of these casted votes, 16 different features are saved, which describe on which decision the voters have based their vote. The data needs little preprocessing as the boolean values are encoded as y,n and unknown and we mapped them to 1,0 and -1. If the preprocessing of unknown values to -1 is useful is debatable, but it seems to work quite fine.

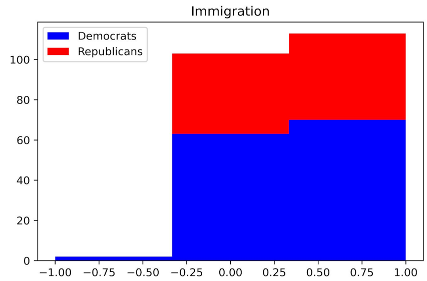


As all of these features describe political opinions in the US, we expect to have clear sides on most of these questions, as both parties have a course they follow. We do not have a correct interpretation for all of this data, we tried to interpret them by ourselves.

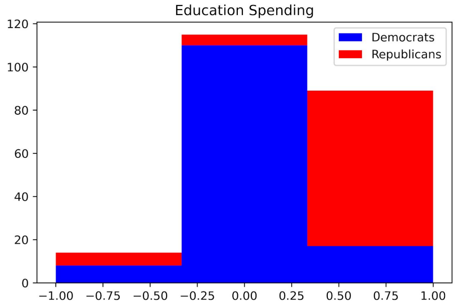
If we look for example at the religious groups in schools, nearly all republicans are for them, whereas the Democrats are split about the idea of having religious groups in schools.



When looking at the immigration statistics this does not represent such a clear picture as the opinions pro and against immigration are split in the parties as well. About half of both parties are for immigration and the other half is against it.

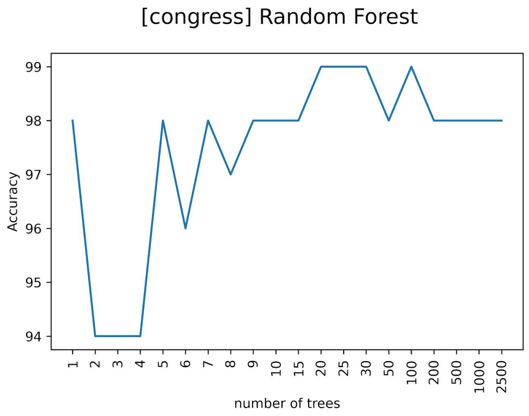


In the last step we want to look at the distribution of the opinion about educational spending, here we can see that most democrats answered this question with no, whereas most republicans answered it with yes. A possible question which could have been asked is, if the education should be financed by the people themselves and not the state.



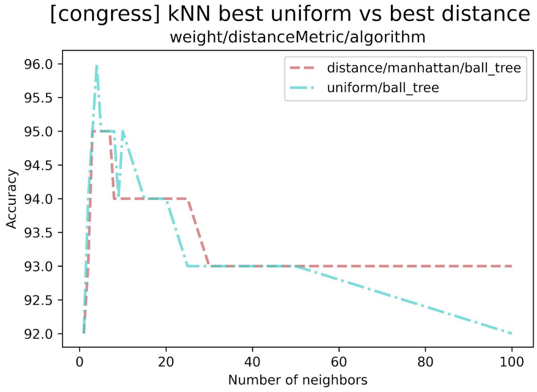
For this dataset it would have been great, to have an explanation to all of the different features, as without these features it is hard to analyze the set. Clearly we can run tests on it, but it is not really feasible for us to reconstruct the questions, which have been answered.

Classification Results and Evaluation



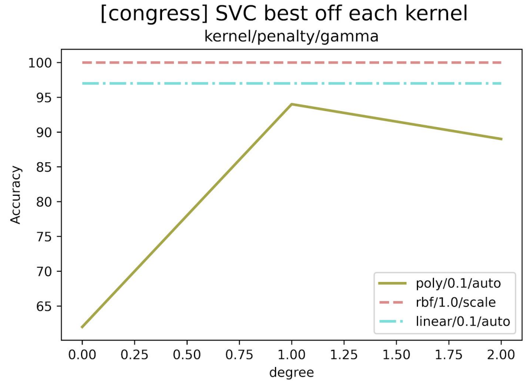
Random Forest

The random forest approach can deliver good results with the zoo dataset, but they tend to vary much. This is due to the small test set we have of only 31 entries. Starting with 1 tree, the algorithm could score 98% accuracy, but with 2 to 4 trees only 94% of the predictions were right. 4 trees again deliver a very good result. These variations with so little trees are so different due to the randomness of this approach, at 10 trees the predictions become more robust and stays at 98% accuracy and above.



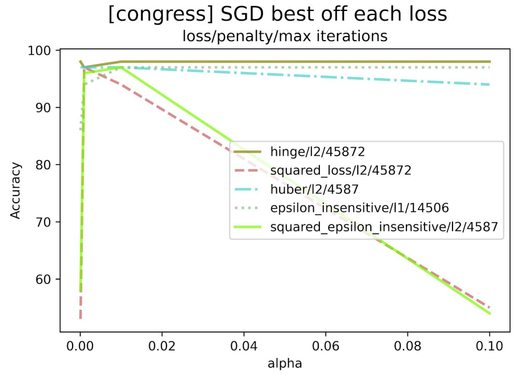
K-Nearest Neighbours

The k-nearest neighbor's approach also gives pretty good predictions, but only for a little number of neighbors. We again analyzed combinations of distance metrics and algorithms and took the best resulting combination of each weight. For the "distance" weight the best distance Metric was the manhattan distance metric combined with the ball_tree algorithm. This approach could achieve 96% accuracy with just 4 neighbors. The more neighbors were added later, the worse the result got for both uniform and distance.
With "uniform" weight, the results were overall pretty similar, just the peak was not that steep and the result was slightly worse.



Support Vector Machine

The support vector machine approach delivers both okay and really bad results. Again, combinations of kernel, penalty, and gamma are shown. The accuracy is plotted against the degree of the function. The linear and rbf approach are both producing very good accuracy, with the linear model being at 96% and the rbf model at 100%. The poly approach does not deliver consistently good results with only 62% accuracy for 0 degrees. With 1 degree the accuracy of the polynomial approach rockets up to 94%, but descends with 2 degrees. All runs performed best with 0.1 penalty and gamma set to "auto", just the rbf works better with gamma set to "scale".



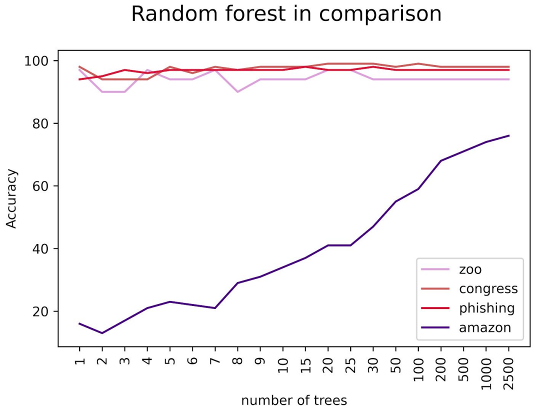
Stochastic Gradient Descent

The stochastic gradient performs best with a hinge loss and 45872 iterations. A combination of loss, penalty and max iterations was used. The accuracy was plotted against the alpha value of the function. The combinations behave very differently on the variation of alpha. The all time high of over 97% could be achieved with hinge loss, l2 penalty and 45872 max iterations. With increasing alpha also the accuracy of our results increase, but at alpha 0.01 alpha starts decreasing. Especially for the squared and the squared_epsilon loss we see a rapidly falling accuracy level. The other two options (huber and epsilon_insensitive) remain nearly on the same level of accuracy as the hinge function.

These values were evaluated on our 99% accuracy prediction.

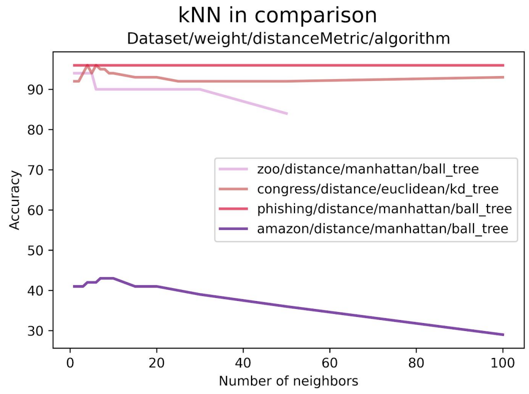
Conclusion

Random Forest



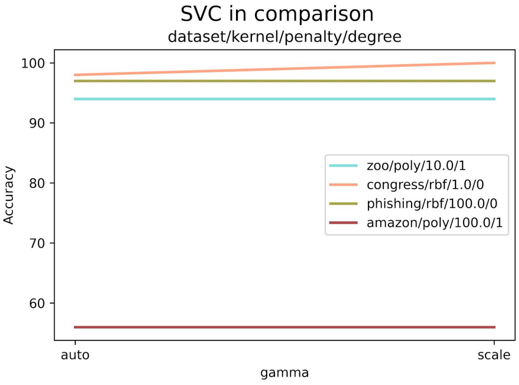
As seen in the chart, comparing all datasets Random forest performs reasonably well across all datasets, even with Amazon Reviews, which is rather challenging on its own, we get reasonably good results given a large enough number of trees. On our smaller datasets, we achieve close to optimal performance with less than 25 trees. Overall the performance of random forest is probably the most reliable across all that we have reviewed.
The calculation of 2500 trees for the phishing dataset took only 15 seconds, whereas for the amazon dataset, the calculations were only finished after 38 minutes.

K-Nearest Neighbours



K-Nearest Neighbours is performing well for all datasets but Amazon reviews. This is due to the fact that Amazon Reviews is both very large, sparse, and has a lot of features. kNN naturally performs best, we are using a small k, since making k too large takes too many neighbors into account which dilutes any results. The plot with zoo ending at around 70 is due to the dataset itself being so small. Across the board, results get worse results as soon as the number of neighbors gets larger than 5. For distance function, we choose to represent the best for each dataset, which in most cases was manhattan distance, though this is less significant since a lot of our features are binary and classes instead of numeric.
Per combination of parameters, the computation time was about 2 seconds for the phishing dataset and 8 seconds for the amazon dataset.

Support Vector Machine

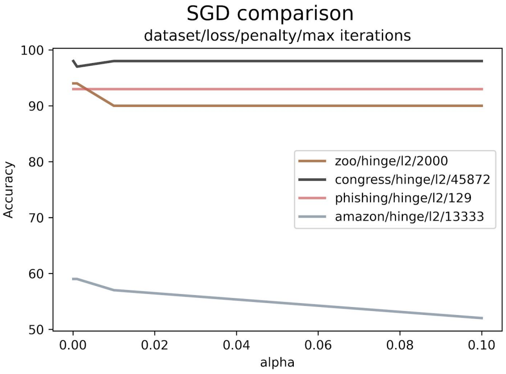


We tried different kernels for Support Vector Machine, where consistently RBF (Radial Basis Function) performed best for our dataset, while linear as a kernel performed the worst. While we used our framework scikit-learn to determine the gamma for our RBF based SVC, here is a comparison between:

- if gamma='scale' is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if ‘auto’, uses 1 / n_features.

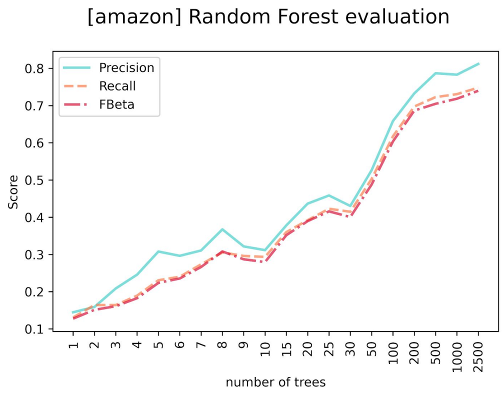
Overall SVM with the right parameters performs well form most of our datasets, except for Amazon reviews, though using SVM with a linear kernel is outperformed by most of the algorithms presented. The computation time can take multiple hours per run.

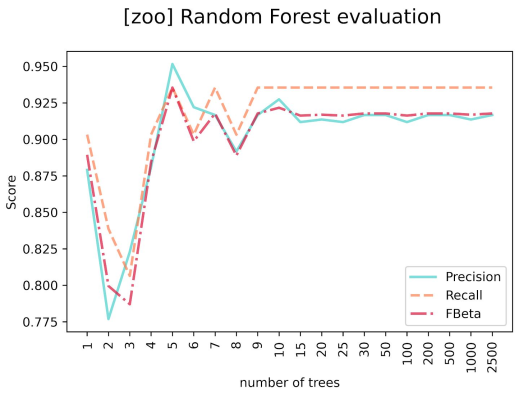
Stochastic Gradient Descent



Stochastic Gradient Descent is kind of a mixed bag when it comes to performance. It is able to outperform other algorithms to some degree, but only in its optimal best case. It requires a lot of finetuning and other than for example random forest or kNN, there does not seem to be a clear rule of thumb on at which point we get better or start getting worse results. Also, it proves to be the worst in terms of computational performance. At times it needs multiple hours per run.

Comparison of different metrics

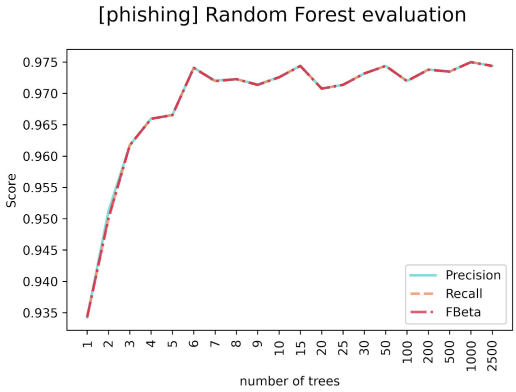


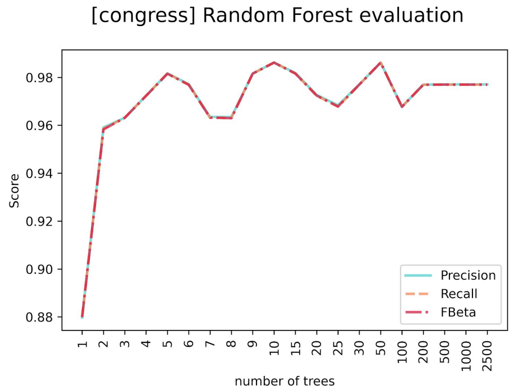


In terms of metrics, we went with accuracy to evaluate the results of our algorithms and for plotting. This was mostly done for simplicity. Accuracy is also the most intuitive one, so a new student trying to identify the quality of an algorithm might go about defining a fraction of the number of true positives and true negatives divided by the total number of samples as a metric for the performance of an algorithm.

There are of course a number of other metrics that can be used for evaluation. In comparison, we used our results of random forest on the dataset zoo to illustrate that in some cases those are very similar and in others can give vastly different results.

If there is a great imbalance of classes, for example, we have binary results, wherein 99% of our dataset would be X and 1% would be Y, accuracy would be a bad metric since a classifier tagging all samples as X would get a 99% accuracy, while in reality performing quite poorly.





Lessons Learned

As our datasets were mostly preprocessed, it would be interesting to pick datasets that may need a larger portion of preprocessing, as this data cleaning is an important and complex part of using machine learning algorithms.

Also, did we realize that the zoo dataset is not really interesting for classification, as there are only 100 entries, which we can learn from. Since the exercise asked for a diverse set of datasets we used this as a baseline for learning on a very small dataset, although most if not all machine learning algorithms tend to perform better on large sets. We have already described several more interesting datasets, which can be based on animals.

Since we weren’t too familiar with the framework to start with, we implemented accuracy as a metric ourselves instead of going with scikit learns builtin metric functions. Since metrics are important and making a mistake implementing a metric can lead to skewed results, we are going to rely on the libraries’ metrics going forward.