## Dataset:

1. Chess (King-Rook vs. King)

* **url** : <https://archive.ics.uci.edu/dataset/23/chess+king+rook+vs+king>
* **github :** <https://github.com/JayeshAbhishek/Chess--King-Rook_vs_King>
* <https://github.com/yaswanthpothuru/Chess-Endgames/blob/master/Report.pdf>
* <https://github.com/pratibha-gupta/Chess_Endgame/blob/master/CHESS.ipynb>
* <https://github.com/nikolagavranovic/chess-possitions-classification/blob/main/king_and_rook_vs_king_classification.ipynb>
* **description** :

An Inductive Logic Programming (ILP) or relational learning framework is assumed (Muggleton, 1992). The learning system is provided with examples of chess positions described only by the coordinates of the pieces on the board. Background knowledge in the form of row and column differences is also supplied. The relations necessary to form a correct and concise classifier for the target concept must be discovered by the learning system (the examples already provide a complete extensional definition). The task is closely related to Quinlan's (1983) application of ID3 to classify White King and Rook against Black King and Knight (KRKN) positions as lost 2-ply or lost 3-ply. The framework is similar in that the example positions supply only low-grade data. An important difference is that additional background predicates of the kind supplied in the KRKN study via hand-crafted attributes are not provided for this KRK domain.

Chess endgames are complex domains which are enumerable. Endgame databases are tables of stored game-theoretic values for the enumerated elements (legal positions) of the domain. The game-theoretic values stored denote whether or not positions are won for either side, or include also the depth of win (number of moves) assuming minimax-optimal play. From the point of view of experiments on computer induction such databases provide not only a source of examples but also an oracle (Roycroft, 1986) for testing induced rules. However a chess endgame database differs from, say, a relational database containing details of parts and suppliers in the following important respect. The combinatorics of computing the required game-theoretic values for individual position entries independently would be prohibitive. Therefore all the database entries are generated in a single iterative process using the ``standard backup'' algorithm (Thompson, 1986).

A KRK database was described by Clarke (1977). The current database was described and used for machine learning experiments in Bain (1992; 1994). It should be noted that our database is not guaranteed correct, but the class distribution is the same as Clarke's database. In (Bain 1992; 1994) the task was classification of positions in the database as won for white in a fixed number of moves, assuming optimal play by both sides. The problem was structured into separate sub-problems by depth-of-win ordered draw, zero, one, ..., sixteen. When learning depth d all examples at depths > d are used as negatives. Quinlan (1994) applied Foil to learn a complete and correct solution for this task.

The typical complexity of induced classifiers in this domain suggest that the task is demanding when background knowledge is restricted.

* **Attributes**
  + 1. White King file (column)
  + 2. White King rank (row)
  + 3. White Rook file
  + 4. White Rook rank
  + 5. Black King file
  + 6. Black King rank
  + 7. optimal depth-of-win for White in 0 to 16 moves, otherwise drawn {draw, zero, one, two, ..., sixteen}.

1. Occupancy Detection:

* URL : <https://archive.ics.uci.edu/dataset/357/occupancy+detection>
* Github : <https://github.com/LuisM78/Occupancy-detection-data>
* <https://github.com/galib96/occupancy-detection-uci-data/blob/main/CODE_FILES/occupancy_detection.ipynb>
* <https://github.com/pcko1/occupancy-detection>
* <https://github.com/shayanalibhatti/Predicting_room_occupancy_using_logistic_regression>
* https://github.com/Rhuax/Occupancy\_detection
* **Description**: Experimental data used for binary classification (room occupancy) from Temperature,Humidity,Light and CO2. Ground-truth occupancy was obtained from time stamped pictures that were taken every minute.

Three data sets are submitted, for training and testing. Ground-truth occupancy was obtained from time stamped pictures that were taken every minute.

For the journal publication, the processing R scripts can be found in:

<https://github.com/LuisM78/Occupancy-detection-data>

* Attributes:
  + date time year-month-day hour:minute:second
  + Temperature, in Celsius
  + Relative Humidity, %
  + Light, in Lux
  + CO2, in ppm
  + Humidity Ratio, Derived quantity from temperature and relative humidity, in kgwater-vapor/kg-air
  + Occupancy, 0 or 1, 0 for not occupied, 1 for occupied status

# Dataset definition:

**A description of your classification problems, and why you feel that they are interesting. Think hard about this. To be at all interesting the problems should be non-trivial on the one hand, but capable of admitting comparisons and analysis of the various algorithms on the other. Avoid the mistake of working on the largest most complicated and messy dataset you can find. The key is to be interesting and clear, no points for hairy and complex.**

Dataset 1 : King-Rook vs. King

# Model results and analysis :

Why did you get the results you did?

Compare and contrast the different algorithms.

What sort of changes might you make to each of those algorithms to improve performance?

How fast were they in terms of wall clock time? Iterations?

Would cross validation help (and if it would, why didn't you implement it?)?

How much performance was due to the problems you chose?

How about the values you choose for learning rates, stopping criteria, pruning methods, and so forth (and why doesn't your analysis show results for the different values you chose? Please do look at more than one. And please make sure you understand it, it only counts if the results are meaningful)?

Which algorithm performed best? How do you define best?

What are the basic things needed for the above?

### Training, cross validation and testing error rates

1. **LEARNING CURVES**

* performance on both training and test data as a function of **training size**
  + A graph and text on a white background

    Description automatically generated
  + **bias and variance / overfit and underfit**
  + *minimum* amount of samples needed to learn effectively
  + A drawing of a curve

    Description automatically generated with medium confidence
* performance on both training and test data as a function of **training times /iteration (loss curve)**
  + **Required for NN : may help learning rate issues,** gradient saturation, or anomalies within your data.
  + A graph and diagram of a good and x

    Description automatically generated with medium confidence

1. **Validation CURVES ( study of each hyper parameter separately**
   1. bias and variance trade-off but with hyperparameters
   2. A graph of a graph of a graph

      Description automatically generated with medium confidence
   3. A close-up of a white background

      Description automatically generated

* end accuracies
* **“reproducible code” : seeds**
* **Can produce tabular data**

**Note :**

* **intermediate learning and validation curves** that show sub-optimal performance and therefore instances of underfit / overfit are likely to be very useful to your analysis

**Hyper parameters :** *at least two* for each algorithm

* **DT:** Pruning + 1
* **Boosting:** # of Weak Learners + 1
* **NN:** Hidden Layer Size (Width, Depth) +1
* **SVM:** Kernel Type +1
* **KNN:** K +1

**Dataset : check for imbalance classes**

Note : They link the results back to some algorithmic behavior, hyperparameter interaction with a given algorithm, and their input data.

**Headings in the papers:**

1. Introduce your data. Talk about why it's interesting (from an ML POV) and what preprocessing you did (be brief).
2. Talk about your experimental methodology.
3. Discuss results isolated to each algorithm.
4. Compare and contrast results across algorithms, across data sets.
5. A solid conclusion to wrap up your journey. Tie in some stuff you learned from lecture that applied to the assignment and more!
6. Introduce your data. Talk about why it's interesting (from an ML POV) and what preprocessing you did (be brief).

Two datasets were selected for classification problems.

1. **King-Rook vs. King :**

- [ ] Datasets

- [ ] Select two datasets

- [ ] Preprocess datasets

- [ ] Choose a performance metric (accuracy, error, precision or recall) for each dataset and justify choice

- [ ] Introduce each dataset and write why they are interesting, and brief description of the preprocessing

- [ ] Talk about overall experimental methodology

- [ ] Decision Trees (DT)

- [ ] Learning Curve plot and analysis

- [ ] Keep the optimal hyperparameter choice . Vary the training data set size , train models with data and plot the curves

- [ ] bias and variance / overfit and underfit analysis

- [ ] Validation curves for at least two hyperparameters

- [ ] Pruning is a required hyperparameter

- [ ] Vary hyperparameter of interest keeping all other optimal hyperparameters fixed and train the models and plot the curves

- [ ] bias and variance / overfit and underfit analysis

- [ ] Add intermidate learning or validation curve to show suboptimal performance and analysis

- [ ] Link back results to some algorithmic behavior, hyperparameter interaction between the algorithm and input data, etc

- [ ] Document wall clock times

- [ ] Tune model to be optimal (GridSearch allowed)

- [ ] Analyze results isolated to other algoritms

- [ ] Compare and contrast results across algorithms and across datasets

- [ ] Neural Networks (NN)

- [ ] Learning Curve plot and analysis

- [ ] X axis should be iterations or epoch - ie loss curve

- [ ] Keep the optimal hyperparameter choice . Vary the training data set size , train models with data and plot the curves

- [ ] Bias and variance / overfit and underfit analysis

- [ ] Validation curves for at least two hyperparameters

- [ ] Hidden Layer Size is a required hyperparameter

- [ ] Vary hyperparameter of interest keeping all other optimal hyperparameters fixed and train the models and plot the curves

- [ ] Bias and variance / overfit and underfit analysis

- [ ] Add intermidate learning or validation curve to show suboptimal performance and analysis

- [ ] Link back results to some algorithmic behavior, hyperparameter interaction between the algorithm and input data, etc

- [ ] Document wall clock times

- [ ] Tune model to be optimal (GridSearch allowed)

- [ ] Analyze results isolated to other algoritms

- [ ] Compare and contrast results across algorithms and across datasets

- [ ] Boosting

- [ ] Learning Curve plot and analysis

- [ ] Keep the optimal hyperparameter choice . Vary the training data set size , train models with data and plot the curves

- [ ] Bias and variance / overfit and underfit analysis

- [ ] Validation curves for at least two hyperparameters

- [ ] \# of Weak Learners is a required hyperparameter

- [ ] Vary hyperparameter of interest keeping all other optimal hyperparameters fixed and train the models and plot the curves

- [ ] Bias and variance / overfit and underfit analysis

- [ ] Add intermidate learning or validation curve to show suboptimal performance and analysis

- [ ] Link back results to some algorithmic behavior, hyperparameter interaction between the algorithm and input data, etc

- [ ] Document wall clock times

- [ ] Tune model to be optimal (GridSearch allowed)

- [ ] Analyze results isolated to other algoritms

- [ ] Compare and contrast results across algorithms and across datasets

- [ ] Support vector machines (SVM)

- [ ] Learning Curve plot and analysis

- [ ] Keep the optimal hyperparameter choice . Vary the training data set size , train models with data and plot the curves

- [ ] Bias and variance / overfit and underfit analysis

- [ ] Validation curves for at least two hyperparameters

- [ ] Kernel Type is a required hyperparameter

- [ ] Vary hyperparameter of interest keeping all other optimal hyperparameters fixed and train the models and plot the curves

- [ ] Bias and variance / overfit and underfit analysis

- [ ] Add intermidate learning or validation curve to show suboptimal performance and analysis

- [ ] Link back results to some algorithmic behavior, hyperparameter interaction between the algorithm and input data, etc

- [ ] Document wall clock times

- [ ] Tune model to be optimal (GridSearch allowed)

- [ ] Analyze results isolated to other algoritms

- [ ] Compare and contrast results across algorithms and across datasets

- [ ] K-Nearest Neighbor (kNN)

- [ ] Learning Curve plot and analysis

- [ ] Keep the optimal hyperparameter choice . Vary the training data set size , train models with data and plot the curves

- [ ] Bias and variance / overfit and underfit analysis

- [ ] Validation curves for at least two hyperparameters

- [ ] K is a required hyperparameter

- [ ] Vary hyperparameter of interest keeping all other optimal hyperparameters fixed and train the models and plot the curves

- [ ] Bias and variance / overfit and underfit analysis

- [ ] Add intermidate learning or validation curve to show suboptimal performance and analysis

- [ ] Link back results to some algorithmic behavior, hyperparameter interaction between the algorithm and input data, etc

- [ ] Document wall clock times

- [ ] Tune model to be optimal (GridSearch allowed)

- [ ] Analyze results isolated to other algoritms

- [ ] Compare and contrast results across algorithms and across datasets

- [ ] Write solid conclusion

New data :

1. <https://www.kaggle.com/datasets/adityakadiwal/water-potability>
2. <https://www.kaggle.com/datasets/shrutimechlearn/churn-modelling>
3. <https://www.kaggle.com/datasets/yasserh/wine-quality-dataset>
4. <https://www.kaggle.com/datasets/rajyellow46/wine-quality>
5. <https://www.kaggle.com/datasets/elakiricoder/gender-classification-dataset>

Section 1. Introduction

Section 2. Why is the data interesting:

Data 1 : Wine quality

Describing data from a machine learning point of view for an assignment typically involves providing essential information about the dataset you are working with. When describing data for machine learning, you should consider the following key aspects:

1. \*\*Dataset Overview\*\*:

- Begin by introducing the dataset. Mention the dataset's name, source, and any relevant background information about its origin and purpose.

Wine Quality dataset, describes the psychochmecal and sensor readings of Portuguese "Vinho Verde" wine [], used for the classification of the wine quality downloaded from Kaggle, <https://www.kaggle.com/datasets/rajyellow46/wine-quality>

2. \*\*Data Size\*\*:

- Indicate the size of the dataset in terms of the number of samples (rows) and features (columns).

\*\*Data Types\*\*:

- Describe the types of data present in the dataset. Mention whether the data is numeric (continuous or discrete), categorical, or a combination of both.

\*\*Feature Description\*\*:

- Provide a list of the dataset's features (columns) along with brief descriptions of each feature. Include information about the data type, units (if applicable), and the role of each feature in the context of the problem.

The data contains 6463 records post cleaning and 12 features, namely [type, fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol], of which predominantly (11) features are continuous and 1 (type) is categorical with 2 categories.

5. \*\*Data Distribution\*\*:

- Discuss the distribution of the target variable(s) if it's a supervised learning task. Is it a classification problem with multiple classes or a regression problem with continuous values? Describe the class distribution if it's a classification problem.

The target variable, quality is ordinal, ranging from 3 to 9, where higher is better quality. The distribution across quality is unbalanced, where 5,6 and 7 corresponds to 32,46 and 16% of the total distribution and remaining 4 categories representing less than 3% each.

This could result in major prediction errors for the cases on both tails, and therefore a derived target feature quality\_cat (average or better) corresponding to quality >5 is created, resulting in 2 categories, of distribution 36.7 and 63.3 for low and average-or-better respectively,

6. \*\*Missing Data\*\*:

- Analyze and report if there are any missing values in the dataset. Explain how missing values are handled, whether they are imputed or removed.

Rows with no-missing values corresponded to 99.5% of the total data and therefore missing value rows were dropped, without losing any generalization.

7. \*\*Data Preprocessing\*\*:

- If any preprocessing steps have been applied to the data, such as feature scaling, encoding categorical variables, or dimensionality reduction, describe these steps and their rationale.

Singe “type” and “quality\_cat” corresponded to 2 categories each, both were converted to Boolean variable, where “white” and “average or above” representing 1 respectively.

8. \*\*Data Visualization\*\*:

- Include visualizations of the data, such as histograms, scatter plots, or box plots, to illustrate key characteristics, relationships, or anomalies in the data.

. \*\*Correlations and Relationships\*\*:

- Discuss any observed correlations or relationships between features and the target variable(s). Use correlation matrices or scatter plots to support your observations.

Fig [1] shows the univariate distribution of all the features including target variable, as seen the data 8 out of 11 numeric features show high skewness, except “fixed acidity, Ph and alcohol. Each model may have different prediction sensitivity to different kind of distribution and therefore would be interesting to study.

Based on Fig[2] there are no strong correlations observed in the data, except of with total sulfer dioxide with free sulfer dioxide and type of -0.72 and 0.7 respectively, however, were not treated for this study.

12. \*\*Train-Test Split\*\*:

- If applicable, explain how the dataset is split into training and testing subsets for machine learning model evaluation. Specify the ratio or method used for splitting.

The data is divided with a proportion of 80:20 in train and test set, where training data is further used to produce cross validation set separate from the true training set..

13. \*\*Evaluation Metric\*\*:

- Mention the evaluation metric(s) that will be used to assess the performance of machine learning models on this dataset. Explain why these metrics are chosen and their relevance to the problem.

For the purpose of our study we use Negative log loss as an evaluation metric. Log loss Measures the dissimilarity between the predicted probabilities and the true class labels. Minimizing it corresponds to maximizing the likelihood of the true labels given the predicted probabilities

Log loss is sensitive to the correctness of individual predictions. It penalizes more heavily for high-confidence incorrect predictions. In other words, if the model is confident about an incorrect prediction, log loss will be high. This sensitivity encourages the model to make well-calibrated probability estimates, which in turn handles any class imbalance in the data.

14. \*\*Challenges and Considerations\*\*:

- Discuss any challenges or considerations specific to the dataset that may impact the choice of machine learning algorithms, model evaluation, or preprocessing steps.

Data 2 : Gender Prediction

Describing data from a machine learning point of view for an assignment typically involves providing essential information about the dataset you are working with. When describing data for machine learning, you should consider the following key aspects:

1. \*\*Dataset Overview\*\*:

- Begin by introducing the dataset. Mention the dataset's name, source, and any relevant background information about its origin and purpose.

Gender classification dataset, is a synthetic data, downloaded from Kaggle, describing various aspects a human faces, which can be used to predict its gender.

2. \*\*Data Size\*\*:

- Indicate the size of the dataset in terms of the number of samples (rows) and features (columns).

\*\*Data Types\*\*:

- Describe the types of data present in the dataset. Mention whether the data is numeric (continuous or discrete), categorical, or a combination of both.

\*\*Feature Description\*\*:

- Provide a list of the dataset's features (columns) along with brief descriptions of each feature. Include information about the data type, units (if applicable), and the role of each feature in the context of the problem.

The data contains 5001 records with 7 features, namely ['long\_hair', 'forehead\_width\_cm', 'forehead\_height\_cm', 'nose\_wide',

'nose\_long', 'lips\_thin', 'distance\_nose\_to\_lip\_long'], contrary to dataset 1, 5 of these features are categorical in nature with only 2 continuous in nature ('forehead\_width\_cm', 'forehead\_height\_cm' ). This dataset was selected to observe the difference in the nature of different models while handling two different kind of features set.

5. \*\*Data Distribution\*\*:

- Discuss the distribution of the target variable(s) if it's a supervised learning task. Is it a classification problem with multiple classes or a regression problem with continuous values? Describe the class distribution if it's a classification problem.

The target value as defined above is categorical, with 2 possible values, Male and Female, equally dividing (50%:50%) the records into 2 halves. This is additional factor of observation, as the 1st data is still slightly skewed in nature.

6. \*\*Missing Data\*\*:

- Analyze and report if there are any missing values in the dataset. Explain how missing values are handled, whether they are imputed or removed.

There is no missing data.

7. \*\*Data Preprocessing\*\*:

- If any preprocessing steps have been applied to the data, such as feature scaling, encoding categorical variables, or dimensionality reduction, describe these steps and their rationale.

The data was not processed separately.

8. \*\*Data Visualization\*\*:

- Include visualizations of the data, such as histograms, scatter plots, or box plots, to illustrate key characteristics, relationships, or anomalies in the data.

. \*\*Correlations and Relationships\*\*:

- Discuss any observed correlations or relationships between features and the target variable(s). Use correlation matrices or scatter plots to support your observations.

As shown in Table/Fig [1] the distribution for all but 1 categorical feature ‘long\_hair” are symmetric, additionally, 2 variables, show balanced data in lower 75% of the values, with a light skew on the higher end, that is the distributions are not completely uniform.

This showcase a much more balanced data, avoiding possible model deteriorations because of data skewness

As seen, in the fig(2), the degree of correlation between features varies, with no inverse proportionality. And significant proportionality among features associated with nose and lips and also gender,

12. \*\*Train-Test Split\*\*:

- If applicable, explain how the dataset is split into training and testing subsets for machine learning model evaluation. Specify the ratio or method used for splitting.

Training, test and validations sets are created similar to dataset 1, for better understandability of the models.

**Modelling methodology:**

* Each algorithm:
  + Describe algorithm and parameters explored
  + Result per data 1
    - Loss curve
      * Overfitting underfitting
      * Pref and inductive bias
    - Learning curve
  + Result for data 2
    - Loss curve
      * Overfitting underfitting
      * Pref and inductive bias
    - Learning curve
      * Overfitting underfitting
      * Pref and inductive bias
  + Compare results between data
    - Loss curve
      * Overfitting underfitting
      * Pref and inductive bias
    - Learning curve
      * Overfitting underfitting
      * Pref and inductive bias
* For each data
  + Data 1
    - Compare performance of all algorithms
      * Loss curve
        + Overfitting underfitting
        + Pref and inductive bias
      * Learning curve
        + Overfitting underfitting
        + Pref and inductive bias
  + Data 2
    - Compare performance of all algorithms
      * Loss curve
        + Overfitting underfitting
        + Pref and inductive bias
      * Learning curve
        + Overfitting underfitting
        + Pref and inductive bias
* Give best algo for each data
* Give best algo overall

**Section 3 : Modelling and analysis methodology** :

5 algorithms were modelled over the each dataset, where the nature of the underfitting and overfitting was studied along with preference and inductive bias of the model across different parameter. Best set of parameters were found using iterative validating ( grid search), in a limited scope of parameter set and range. The learning curve of the each model for best found and a suboptimal variant was further studied.

The analysis in the following sections are segmented into 3 segments:

1. Analysis of the each algorithm, across different datasets.

2. Comparitive analysis across algorithms for each dataset.

3. Summary analysis.

**Section 4 :** Analysis of the each algorithm, across different datasets

This section covers a detailed analysis of the implementation of 5 different algorithm mentioned in section 1, applied to datasets as discussed in section 3.

link the results back to some algorithmic behavior, hyperparameter interaction with a given algorithm, and their input data

What not to do :

* Summarizing rather than analyzing your results.
* Rotely describing how the algorithms work in some introduction section.
  1. ***Decision Tree***

Decision tree was trained and optimized by pre-pruning using Max\_depth and post-pruning using Cpp-alpha (cost complexity parameter) over both the datasets, as detailed below.

**Loss curve**

A screenshot of a graph

Description automatically generated

A graph of a graph

Description automatically generated with medium confidence

**Parameter 1 : cost complexity parameter – ccp\_alpha**

* **Dataset 1:**
* At smaller values the -ve log loss (NLL) for training data is observed very high, where the NLL for validation set was very low, representing low classification accuracy, representing high overfitting, possible reason. However, with increase in value of the parameters, both train and validation NLL move closer reaching a peak NLL value of ~ -0.533 for CV, at ~ 0.00265, reaching highest generalization performance. With further increase in the parameters both train and CV NLL start to gradually decrease together, moving towards underfitting,
* CCP\_Alpha controls the trade-off between complexity and accuracy, for DT, complexity represents number of leaf nodes, At lower ccp-alpha the pruning is limited resulting in deeper (more complex) tree, fitting the training instances more accurately, however due to the preference bias of DT the lower nodes constitute to very less information gain as compared to as nodes root node, resulting fitting models to training data specific intricacies, creating loss of generalization over unseen instance, as observed in low NLL for CV and high NLL for training data. As most data features for data 1 are continuous in nature, it creates opportunity for creating more defined rules/cuts over the features and a higher chance of overfitting
* Similarly, at high values of ccp-alpha, more number of nodes trimmed are closer to root node, corresponding to higher information gain, these nodes may represent minimum specificity required for representing part of target function and loosing which results in moving towards more general rule creating nodes, effectively decreasing both NLL training and CV.
* **Dataset 2:**
* Like dataset 1, the decision tree model overfits at lower values of ccp\_alpha, reaches its peak of NLL ~ -0.1066 at ~ 0.0008 and then declines to start underfitting as the parameter value increases. Compared to dataset 1, this set shows significantly better classification performance at the peak, this is true for the lowest and highest complexity of the model for data 2 as well, Additionally the peak performance is attained at much lower parameter value for this dataset.
* Reason : less parameters in dataset 2 therefore max depth is less, additionally most features are categorical in nature with only two possible values further limiting the possible cuts which can be made, describing early reaching of the peak.
* Additionally as the data is synthetically created, from the correlation charts, more features have higher correlation with the target feature as compared to the dataset 1, which could result in significant increase in information gain.
* A common observation for both the dataset is after certain value of ccp-alpha, the decent toward underfit is much gradual, meaning these parameter values values were not enough to remove any highly significant node yet, therefore these nodes would be much closed to the root, or could be the root node itself.

**Parameter 2 : Max -depth**

* **Dataset 1:**
* Higher max-depth represents a more complex tree and similar to ccp-alpha at lower values, therefore the loss curve for the model, as expected, shows underfit at lower depth values, that is CV and Train NLL are low, which attains a peak NLL of ~ -0.5319 for cv at depth 4, as postulated for ccp-alpha, the performance difference between depth 1 and peak performance depth 4 is not much, that is information gain attainted by root node is significantly high compared to the subsequent feature rules, supporting the slow descent to underfitting for higher values of cpp-alpha for dataset 1. This showcase the preference bias of DT. The peak performance for both the parameter are comparable, which relay both pruning mechanisms are equally good for the current classification problem data. At higher values, the model descents to overfit
* **Dataset 2:**
* DT max-depth parameters behaves similar to Dataset 1, underfit at low values, peak performance at NLL of ~ -0.1116 for cv at depth 4, followed by a descents to overfitting. Compared to Dataset 1, the root node for D2 itself shows a higher performance, additionally subsequent 3 tree depth add significant information gain, unlike D1, resulting in a better peak NLL, which also support, while more features in D2 were correlated with te target, they were less correlated with each other, therefore adding more individual value in classification.

**Learning curve**

The performance of DT with increase in the number training instances was observed, comparing Best found DT parameters for the data and a suboptimal DT based on the final CV NLL.

**Dataset 1:**

**A graph with orange and blue dots

Description automatically generated with medium confidence**

**A graph of a function

Description automatically generated with medium confidence**

* As shown above the performance of suboptimal DT with ccp-alpha : 0.00001 and depth : 6, a more overfit model, was compared to the best found model with 'ccp\_alpha': 0.0002, 'max\_depth': 4, with Max NLL -0.5319, as seen from above in both case the increase in training data decreases the gap in CV and training data performance, i.e more data brings more generalization, however through out the different training size gap in NLL for CV and train data for the suboptimal model is higher because of significant overfit. Additionally the as the number of decision nodes are more with low ccp-alpha and high tree depth the training time for suboptimal model is higher as well, ~ 0.035 compared to ~0.020 of best model for the whole data.

**Dataset 2:**

**A screenshot of a graph

Description automatically generated**

**A graph of a function

Description automatically generated with medium confidence**

* As shown above the performance of suboptimal DT with 'ccp\_alpha' : 0.002,'max\_depth' : 9, a more overfit model, was compared to the best found model with 'ccp\_alpha': 0.0008, 'max\_depth': 9, with Max NLL -0.1058. Unlike dataset 1, the suboptimal and best models have the same pre-pruning parameter maxdepth 9, where the ccp-alpha for best is much lower, which would result in less trimming of latter nodes. As observed in fig [4] and [5], the suboptimal model converges faster to the to its best performance with increase in data, where as the best model which is much deeper/complex shows overfitting with less training data, unlike dataset 1, however shows a better performances with large training set, as discussed for loss curve, nodes till depth 4 showcase high info gain, however when the dataset is less this generalization is not pronounce and model tends to overfit, its only when sufficient data is present the model could find better rules for generalization. Additionally in dataset 1, unlike data 2, the best model converges faster, on the same logic as defined above, the info gain with deeper nodes is not comparatively high from root node, and even with low data the model could find the root node generalization.
  1. ***Adaboost Decision tree***

Adaboost was applied using significantly underfit decision tree for dataset 1 and dataset 2, which was further optimized using number of estimators and learning rate.

**Loss curve :**

**Parameter 1 : n estimators**

**Dataset 1:**

A graph of a graph and a graph of a graph

Description automatically generated

* A base decision tree of max\_depth=1 ,ccp\_alpha=0.0001, which would underfit the data, cpp\_alpha is irrelevant as the node is only 1, boosting with learning\_rate=0.3 which would align more weight to the best weak learner, however would still not avoid other weak learners. As observed above a single estimator was found to produce the best result, NLL ~ -0.5885, this corresponds to DT discussion above, till depth 4, the increase in info gain is not very significant, as the number of estimators increases, stumps which are fitted on other variables are given significant weightage, however these features may not add more value, which results in overall model underfitting. This can be seen by the drastic drop in NLL with increase in estimators. (possible high learning rate)

**Dataset 2 :**

A screenshot of a graph

Description automatically generated

* Dataset 2 shows a very similar behaviour for n-estimators with highest performance at 1,NLL ~ -0.1209, for a base DT of max\_depth=5,ccp\_alpha=0.00005, which from the DT graphs in previous subsection shows would inherit the best tree with depth 4 however with low ccp-alpha would not lose information. Unlike the Dataset 1, the drop in performance with increase in estimators is significantly high, this could be because of more categorical features and limited options to create rules, which would result in significantly underfit stumps.

**Parameter 2 : learning rate LL**

**Dataset 1:**

* The same base DT was used, with 50 estimators, as seen the NLL is high at lower learning rates, giving nearly equal importance to most weak learners, the performances slightly increases with increase in learning rate reaching peak at LL 0.01, NLL -0.5797, where it would be able to sufficiently differential best weak learner vs others. With further increase in learning rate the model starts weighing best weak learners more and prior LL starts to underfit. This could explain the single best estimator in 1st parameter study.
* It can be seen that the performance of boosted DT is lower than single most optimal DT because of this configuration, as even the higher weighted single stumps significantly underfit the data.

**Dataset 2:**

* A very similar observation is made for dataset 2, with same base DT as for previous parameter with 50 estimators. High NLL at lower learning rates, peak at LL 0.03, NLL -0.0779, and underfit at higher values because of reliance on only 1 estimator. The difference in the performance at low and high learning rates for both the datasets is comparable for the same number of estimators.
* However compared to the single DT, the best boosted DT accordingly to LL show improvement, unlike dataset 1, this could be factored to distribution and association of selected features which as they add significant individual information gain unlike dataset 1.

**Learning Curve:**

**Dataset 1 :**

**A graph and a chart

Description automatically generated with medium confidence**

**A graph with numbers and lines

Description automatically generated with medium confidence**

A suboptimal boosted DT with max\_depth=1,ccp\_alpha=0.0001 along with 50 estimators and 0.3 learning rate was used for compared over training set size, with the best found boosted DT constrained to same max\_depth and ccp\_alpha but 'learning\_rate': 0.001, 'n\_estimators': 201, with NLL -0.5788 over whole data, as seen from the fig[8],fig[9] the over all difference in NLL from 10% to whole training data is not significant, that is model performance over CV does not significantly change with increase in training data, however the average performance of best model is higher than the suboptimal model. Which associates the impact of lower learning rate accompanied with more estimators as discussed in the individual analysis.

This consistency in performance at various train sizes reflect easy identification of generalization rules from the data, which reflects the observation of the high NLL with depth 1 and fast converge for DT.

At smaller training set, the NLL of training data is comparatively higher than CV NLL, which indicates relative overfitting, this gap reduces with increase in data.

The overall train time of best boosted tree is 5x of suboptimal model, because of processing 4x number of estimators.

**Dataset 2:**

**A screenshot of a graph

Description automatically generated**

A comparison of a graph

Description automatically generated with medium confidence

A suboptimal boosted DT with max\_depth=5,ccp\_alpha=0.00005 along with 50 estimators and 0.3 learning rate was used for compared over training set size, with the best found boosted DT constrained to same max\_depth and ccp\_alpha but 'learning\_rate': 0.003, 'n\_estimators': 601, with NLL -0.0755 over whole data, as seen from the fig[8],fig[9], similar to dataset 1.

Unlike dataset 1, there is significant difference in NLL over train size for both suboptimal and best model. Where both show high overfitting behaviour at low train size, however this is more pronounce for best model because of high estimator count, as large number of trees easily adjust to intricacies of the training data. Suboptimal model reaches peak faster because of higher learning rate and low number of estimators.

However this change is much more gradual in the best model, as there is more scope for larger number of trees to generalize better over data with smaller learning rate.

Following dataset 1, the training time here as well for best model is high, 12x of suboptimal model over whole data, because of large number of estimators.

* 1. ***K nearest neighbours***

KNN was trained and optimized controlling K (number of neighbours) and p (power of distance) over both the datasets, as detailed below. For this model, no prior knowledge of feature weightage is assumed and therefore to minimize the effect of curve of input features were standardized.

**Loss Curve**

**Parameter 1: K (#** neighbours)

**Dataset 1:**

**A screenshot of a graph

Description automatically generated**

* With base p of 2, using Euclidian distance and uniform weightage, fig[10] shows, estimating class based on less number of neighbours does not generalize well, resulting in high overfit, however, this quickly resolves in only a small increase in #neighbours for estimation, reaching a peak NLL -0.5268 at K=200. with further increase in K, the generalization increases however this also results in loss of local specificity, directing towards underfitting.
* The slow descent into underfitting on the data support the inductive bias for KNN that instances of similar class have similar features, as with the increase in the k does not significantly decrease performance, therefore features of different class must be clustered separately with defined global decision boundaries

**A screenshot of a graph

Description automatically generated**

**Dataset 2:**

Same base model as dataset 1 was used for the study, which showcase as a similar trend of overfitting at low K reaching peak NLL -0.0907 at K=200 and moving towards underfitting as K increases further. However, comparatively the drift towards underfitting is more pronounced unlike dataset 1, supporting there are model local decision boundaries/ clusters compared to data 1. This is further aggregated by the limited number of possible feature values for 6 out of 7 Boolean variables. Additionally, localized impacted is further supported by low difference in lowest NLL K and peak NLL K, compared to dataset 1.

***Parameter 2:* P ( power of distance)**

With a base K = 5 the power of distance was varied for KNN over both the dataset, a small K was taken into consideration towards preference bias of KNN towards localized commonalities.

**Dataset 1:**

As shown in fig [11 b], at small k, the overall CV performance does not change drastically, however KNN attains best CV performance at p= 1, NLL -0.0907, whereas with increase in P to 2 and 3 NLL slightly decrease. As observed form Corelation chart and DT model, only one of the features showed significant association with the target variable, while KNN considers all the features for decision making, at p = 2,3 the difference in distances of irrelevant feature if high, becomes more pronounce, unlick at p =1. With further increase in p the accuracy gradually starts increasing, this could be the result of filtering effect of outliers at high p.

**Dataset 2:**

As fig [12b] shoes , the impact of the p for data 2 is similar over, i.e the change is performance is not immense. The similarity follows for p =1 and 2 as well. However performance jumps back at =3 and gradually decrease from there. Unlike dataset 1, 5 out of 7 features in dataset 2 are Boolean and therefore the difference values at any power they take up is 1 or 0, which is not the case for the 2 continuous features. As seen from fig [2] , these features are only weekly correlated with target, however the impact of the these features may continue to increase at higher powers, resulting in lower accuracy.

**Learning curve:**

A suboptimal KNN with 'k: 5,'p' : 10 was compared over training set size, with the best found KNN for both the datasets.

**Dataset 1:**

**A graph of a function

Description automatically generated with medium confidence**

**A graph of a graph with numbers and lines

Description automatically generated with medium confidence**

Best KNN model with 'k: 100, 'p': 3 , with NLL -0.5238 was found over whole data. As seen in the figure [14] the suboptimal model CV NLL decreases with increase in training data. At smaller as training size the instances are sparse therefore even with lower k and high p, the model is able to generalize well, however with increase in the training data the estimation is more localized resulting in high overfitting. However with the optimal model, where k is very large, the model is too generalized resulting in underfitting as seen in the fig [15], which resolves with the increase in training data. Additionally there is a drastic difference in the CV NLL of suboptimal and best model, at all training sizes, tying back to parameter analysis of k.

**Dataset 2:**

A graph of a graph and a graph of a graph

Description automatically generated with medium confidence

A graph of a graph with numbers and a number of points

Description automatically generated with medium confidence

Best KNN model with ‘k’ : 200, 'p': 1 with NLL -0.0868 was found over whole data. Suboptimal model does not as show any change in the CV NLL over training size, this is because of more Boolean features, limiting the size of concept space, therefore even a relatively smaller dataset may represent the population. There are only two continuous standardised features which could drive the localization effect with increase in data, however this is not visible with p=2 where boolean feature have more weightage due to majority and standardization at a small k of 5. Fig [17] shows a similar best KNN underfitting behaviour as dataset 1 at smaller training size and high k, however this behaviour normalizes faster in dataset 2, because of only 2 continuous variables. Thereafter like suboptimal knn the best model CV NLL does not change much with increase training size, following the same reason. This is unlike dataset 1, where the CV NLL shows significant change throughout the training size, due to more continuous variables, though decelerating at larger sets. The performance of best model is significantly better than suboptimal model, because of better generalization with high k.

* 1. ***Support vector machines:***

SVM classifier was trained and optimized controlling 2 parameters, i.e kernel function and gamma (γ). As SVM is sensitivity to range of feature values, data was scaled before training.

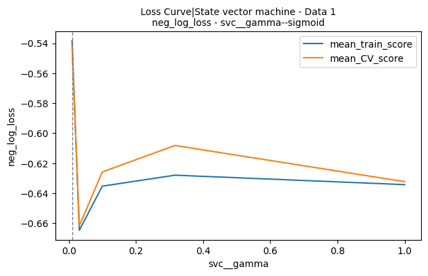
**Loss Curve.**

Due to categorical nature of kernel function, 3 different loss curve corresponding to kernel functions [‘poly’, ‘rbf’ , ‘sigmoid’] were plotted for comparison with different values of gamma for each dataset. For all models beside overall best used in studing the learning curve, regularization term C is kept to 0 to independently study γ.

**Dataset 1.**

A graph of loss curves

Description automatically generated with medium confidenceA graph of loss curves

Description automatically generated

**Polynomial Kernel:**

As observed in fig[19 a] at the model accuracy increase while moving away from γ = 0 and reaches peak NLL of -0.6272 at γ = 0.1 for both CV and train data, with the degree of 3.

This is because even if degree is high, at lower γ (near 0), impact of individual support vectors are less in calculating decision boundary ***w***, which results in underfitting the training data. While here, similarity is a cubic function of the dot product of training instances, this underfitting may also diminish any false classification due to unrelated features. With small increase γ, the impact of individual datapoints increases creating lesser flexible boundaries, resulting in better segregation of training instances. However with further increase in γ a significant drop in CV NLL is visible, which is a result of decision boundary defined too close to training examples, overfitting in the 3rd dimensional projection of the instances, which do not generalize well, additionally the training NLL is also seen to decrease along with CV, this indicates the inability of segregation of data in the projected space, which could be due to the highly skewed nature of continuous features with a strong mix between features values of classes.

**RBF kernel:**

Similar to polynomial kernel, the model accuracy of rbf kernel follow the same pattern of underfitting at low gamma, peak NLL of -0.5183 at γ = 0.1 , best among all 3 kernels, and overfitting at higher gamma values, however is less pronounced, i.e even with narrower and more localized decision boundaries resulting from high c, this could be accounted to two reasons, 1. impact of exponential decay of similarity between instances with distance, creating more localized boundaries to resolve the issue of skewed data feature mix, 2. The projection of data in infinite dimensions, i.e while the data may not be as easy to separate at lower dimensions, at infinite dimensions plains can be found to separate the data, where impact of high γ overfitting could be normalized, the same two reasons support high growth in train NLL as well.

**Sigmoid kernel:**

Compared to both Poly and RBF kernel the difference in NLL is small for both training and CV set. The peak NLL of -0.5418 at γ = 0.01 for both CV and train data. The sigmoid function uses tanh to transform the linear projection of on instance over other, which scales the relations on -1 and 1, the shape of the decision boundary is limited, unlike linear or polynomial kernel. Additionally, it is possible that the data is linearly separable, which may explain the low difference between the performance at low and high γ. Which would also direct towards possible overfit of polynomial kernel with degree 3.

**Dataset 2**

A graph with a line and numbers

Description automatically generated with medium confidenceA graph of a line

Description automatically generated with medium confidenceA graph of loss curves

Description automatically generated

**Polynomial Kernel:**

Similar to Dataset 1, model accuracy drastically increases moving away from γ = 0 and reaches peak NLL of -0.0805 at γ = 0.0316 for both CV and train data, with the degree of 3. Followed by a gradual and slow decreased in the CV NLL but not enough in train data, representing slight overfitting. In addition of the explanation in dataset 1, the drastic initial NLL increase and slowed overfit could be explained by the presence of more Boolean features, that is the number of possible divisions is limited unlike dataset 1, which means decisions boundaries which can classify instances more easily and generalization could be reached faster, therefore with only a small increase in γ a complex enough decision boundary could be identified to segment the classes. Similarly, even if the γ is further increased , although it fits training examples more closely it does not diminish generalization capabilities too much, i.e the data is inherently easy to separate.

**RBF Kernel:**

Like Dataset 1, model accuracy at low γ values is high and slightly increases moving away from γ = 0 and reaches peak NLL of -0.0743 at γ = 0.0316 for both CV and train data, highest among 3 kernels, and transition to overfitting behaviour. However the difference in the NLL for train and CV, throughout the experimented range is very small, therefore RBF even with large γ could segregate classes easily with limited dimensional projections. The performance compared to Polynomial kernel was observed to be better at low and similar to at medium and higher γ, similar to data 1.

**Sigmoid Kernel:**

KNN for sigmoid kernel showcase high NLL throughout γ range compared to dataset 1, however NLL for smaller γ show higher accuracy, reaching a peak NLL of -0.0772 at γ = 0.0316 for both CV and train data, and decreasing monotonically for higher γ for both train and CV sets. Although the range of change across range is small, it is larger than Dataset 1 observations.

**Learning curve:**

The best performance SVC was identified by picking the best model from the parameter analysis based on kernel and γ. This model was further optimized using regularization parameter C for each dataset. A suboptimal SVC was compared over training set size, the best found SVC.

**Dataset 1 :**

**A graph of different sizes and colors

Description automatically generated with medium confidence**

A graph of a diagram

Description automatically generated with medium confidence

Fig [20 a] show the learning curve of suboptimal SVC with γ :0.5,C:0.5, kernel : rbf, compared with Fig[20 b] showing the best SVC at NLL -0.5154, with γ: 0.1, C: 0.1, kernel: rbf. As shown, Train and CV NLL for both suboptimal and best SVC does not change much with as increase in training data. The suboptimal model with high γ and C results in as more overfitted model compared, resulting in large difference in Train and CV NLL which does not minimize drastically with more data, Additionally the best model with small γ and C is more generalized SVC resulting in low and maintained Train NLL, the CV NLL in both the cases are very close throughout, directing towards more impact over segregation of instances with infinite dimension projection, this may points towards equal difficulty in classification even at different bias and variance trade-offs due to highly mixed and skewed features, as seen in the distribution. Additionally this also direct towards easier representation of data even at small samples.

**Dataset 2:**

**A graph and diagram of a graph

Description automatically generated with medium confidence**

**A graph and chart with numbers

Description automatically generated with medium confidence**

the data set suboptimal SVC (γ :0.5,C:0.8, kernel : rbf) and Best SVC (γ : 0.0316,C:1, kernel : rbf, NLL: -0.0741), both show decrease in overfitting with increase in data, only with decrease in train NLL unlike dataset 1, however, like dataset 1, the CV NLL does not value significantly, following the same argument as dataset 1. Best CV NLL compared to suboptimal SVC remains high, where low γ and C creating smoother boundaries, allows the model to avoid highly complex shapes, which may only reduce generalization to a curtain degree because of limited features values. These limited feature values also allow easier finding of separation planes, resulting in high performance in both suboptimal and best model compared to dataset 1.

* 1. ***Neural Networks NN:***

Neural networks

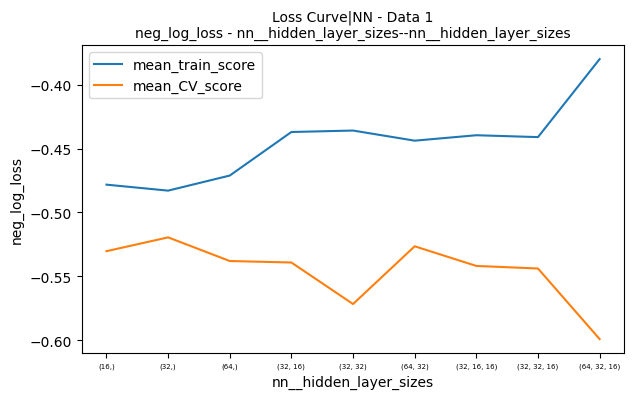
Unlike previous models, multiple parameters including, hidden layer depth and size, learning rate, batch size and momentum, we studied separately for NN over both the datasets, as they all govern the model preferences and restrictions in estimating the target functions. The Loss Curve details 2 of the parameters, hidden layer depth and size, along with batch size in detail. It also briefly covers learning rate as an additional parameter. A base NN model, with two hidden layers , 32-16 with Relu activation was used with initial learning rate 0.001, batch size 24, and momentum 0, using Adam optimizer, for both datasets individually was used varying above mentioned parameters, as detailed below. This base model was identified post iterative study of the learning curve and changing parameters. Change in momentum did not result in major performance changes, and therefore not covered in analysis.

In addition to Learning curve over train size, learning curve over training epoch is also covered, only for best NN models. Lastly as the weight updates are sensitive to feature ranges, all features are scaled before training.

**Loss curve:**

**Parameter 1: Hidden layer shape:**

9 possible hidden layer shapes where compared against, with indicative increase in complexity, 3 single HL (16,), (32,), (64,), 3 two layers HL (32,16,), (32,32,), (64,32,), and 3 3 layer HL (32,16,16,), (32,32,16,), (64,32,16,) . While the a clear order of increase in complexity cannot be accurately quoted, it can be mentioned that deeper layers make use of intermediate features and therefore are more complex, if the number of nodes between shapes are same.

**A graph with blue and orange lines

Description automatically generated**

**Dataset 1:**

Fig [21 a] shows a general trends, with increase in depth as well as breath of the hidden layers, gap between the CV and train NLL increases, This cause is majorly driven by more complex network being able to closely represent training data. However the power of generalization does not monotonically change with this. As seen shallow single layered networks, only capture easily available patterns, on based on the input data, and do not create too many derived intricate features, creating simple decision regions without overfitting the training data, explaining the low difference between the CV and train NLL. For our case these shallow network also have the highest NLL, indicating inherent data separability. The best NN of HL (32,) with NLL -0.5194 indicates, a minor patterns of low NLL at narrower network, because of inability to represent slightly complex relationships, and also at wider network, where more intricate identified patterns overfit the training data. This kind of distinction becomes slightly difficult when the layers are increased. More number of layers result in creation of derived features, which can be used to identify important relationships with target. (32,16) and (64,32) both perform better than (32,32), we can hypothesize that (32,16) compared to (32,32) being simpler create mode generalized classifier, at the same time (64,32) could find more valid derived features and patterns closer to the true target concept, however this may require further validation. The overfitting is more sever at deepest and widest networks. This also is an impact of continuous and skewed data features with a potential to create complex derived features.

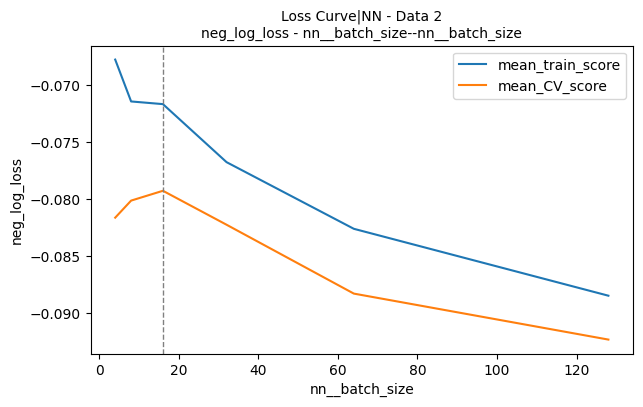
**Dataset 2**

Fig [21 b], follows many of the explanations shared for data 1, however the nature of the highly categorical data, creates differences in the performance. As seen at narrowest and shallowest network, intricate boundaries or derived features are note created, which in this case results in relatively strong underfitting, Although NLL for dataset 2 is very high compared to the best model for data 1, which shows relatively easy class segregations. This is resolved with increasing the width, as seen from corr. Chart features in data 2 show strong association with target, and therefore this increase is enough to identify these linear relationships, where the best NN with HL (64,) with NLL -0.0776 is found. Further increase increase in the complexity via width and depth does not significantly change CV NLL but slightly represent training data better. This is because good relationships could be found in the first layers itself, which may just pass values from one layer to next without major transformations. However this hypothesis was not validated. Additionally more categorical features limit the creation of derived features, therefore the deeper hidden layers may not add value after a curtain important feature derivations are reached.

**Parameter 2 Batch size:**

Depending on the kind of features, their distributions and association of data the generalization power of the batch size varies, therefore different batch sizes were experimented for the datasets.

**A graph with blue and orange lines

Description automatically generated**

**Dataset 1:**

Fig [22 a] shows at smaller batch sizes the model tends to overfit, this is because these result in more number of weight updates with high variance, which could possibly result in larger weights or even reach local minima with suits training data better, which fits the train data better creating more complex decision regions. This also is against the preference bias of simpler networks. This effect reduce with increase in batch size, therefore less w updates. Reaching a peak generalization at 512, NLL -0.5355, a further increase in batch size, can cause high normalization wight changes which slows down the convergence and with insufficient training data or epochs, this may result in underfitting. For our network this is less pronounce, additionally, the CV NLL does not change much throughout Batch size range, possibly due to localized separation of the classes or high mix of classes as seen in the other algorithms.

**Dataset 2:**

The impact of underfitting is more pronounce for dataset 2, unlike data 1, at relatively smaller values. Although the patterns remains same of underfitting at smaller values, peak at 16 with NLL -0.0792, and shift towards underfitting. However, the over-all change in performance is not significant, meaning the segregation function can easily be identified, irrespective to batch size. This is the direct result of limited categorical and balanced variables with significant correlation with target function which also limits the amount of variance which could be introduced with change in batch size.

**Other parameters: initial learning rate**

Initial LR is highly subjective to the association of data with target function and

**A graph with blue line and white text

Description automatically generatedA graph of a graph with blue and orange lines

Description automatically generated**

**Dataset 1**

**Dataset 2**

**Learning Curve – epochs**

Best model was identified for each dataset by performing grid search over all the variables at the same time, in addition with early stopping, and 5 validation sets. A single run of learning curve was created to understand the epoch, it is expected that this may only be indicative of general performance of the model and may vary in case we perform and average this test over different cross validation sets. For this example, the validation NLL was calculated over test split as explained in the data section.

**A graph of a graph

Description automatically generatedA graph of a graph

Description automatically generated**

**Dataset 1:**

Fig [23 a] shows, at less iteration the performance over train and test is similar, for the best NN with 'batch\_size: 512, hidden\_layer\_sizes: (32, 32), LR\_init: 0.00215, momentum: 0, additionally NLL for both these sets is slightly low, which also indicates slight underfitting, however with only 1 more iteration the test NLL peaks to -0.5676, depicting easier identification a model which generalize well, with further iterations, the model fits the training data closely and subsequently follows a slow increase in train NLL, while the test NLL drastically decrease, this indicates fitting to noisy patterns in the training data, coupled with skewed continuous features, adding high variance and very complex and tight decision boundries.

**Dataset 2**

Although dataset 2 follows a similar pattern, with, 'batch\_size: 16, hidden\_layer\_sizes: (64,), LR\_init: 0.03162, momentum:1e-04, with increase in epochs, underfit at less iterations, peak at 130 epochs, NLL -0.0536 and a overfitting phase, the difference change in the performance in not much even at 1000 epochs, this is because of better representation of the target concept in the training data, only two continuous variables, balanced distributions and inherent class segregation, therefore the outliers and the noise in the training data is comparatively low.

**Learning Curve – training size**

The best found NN for each dataset was compared against a suboptimal NN model, over different train sizes and 5 cross validations, similar to other 4 algorithms.

**Dataset 1**

**A graph with orange and blue lines

Description automatically generated**

**A graph with a line and a line graph

Description automatically generated with medium confidence**

Fig [24 a] show the learning curve of suboptimal NN with 'batch\_size': 16, 'hidden\_layer\_sizes': (16,), 'LR\_init': 0.03, 'momentum': 1e-04, compared with Fig[24 b] showing the best NN as specified in the Learning Curve – epochs. Both models were limited to 25 max iterations better comparison. Overall there is a drastic change in performance of suboptimal NN CV NLL, where it remains more stable for the relatively stable for the best NN. This also indicates the training data is representive enough at smaller train size, however, the simpler suboptim network cannot identify these patterns very well, and due to smaller batch size results in overfitting, where best model shows slight indication of underfitting because of very large batch size for the training set and lower learning rate. Which also helps the best NN learn more patterns, given a deeper and wider structure, with slight increase in data, eventually fitting better with the training set as well. More data for suboptimal model also results in better generalization, and decrease in overfitting, resulting increased CV NLL and decreased Train NLL. Large number of updates for suboptimal model also result in higher computational time.

**Dataset 2:**

**A graph and graph chart

Description automatically generated with medium confidence**

**A graph of a graph with a line and a line

Description automatically generated with medium confidence**

Fig [25 a] show the learning curve of suboptimal NN with 'batch\_size': 16, 'hidden\_layer\_sizes': (16,), 'LR\_init': 0.5, 'momentum': 1e-04, compared with Fig[25 b] showing the best NN as specified in the Learning Curve – epochs. The major difference between the both the networks Hidden layer width and learning rate. With increase in the train size both Train and CV NLL for both models move towards each other, in suboptimal this process is more drastic, i.e while with less training example both model overfit, high LR for suboptim result in bigger changes in weights and faster convergence, this also means with further increase in training examples the model is effected by training noises more and also may fail to converge into global minima. Along with narrow hidden layer, it may fail to identify better segregating features, which could result in underfit. On the contrary, smaller learning rate and wider network lets the best model find more intricate features with gradual convergence with small updates. Although this increases training time by 10x

* 1. **Comparison Summary:**

This Sections compares inter data performance and also points to overall differences in learning algorithms between the two datasets.

* 1. **Table:** CV accuracies were noted while identifying the best learners per data and algorithms, which were further used for calculating accuracies for test sets.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Dataset 1** | | | **Dataset 2** | | | |
|  | Best Features | Negative Log Loss | | Best Features | Negative Log Loss | | |
| **Model** | CV (avg of 5) | Test | CV (avg of 5) | | Test |
| Decision Tree | ccp-α:0.0002, depth:4 | -0.5319 | -0.9002 | ccp-α:0.0008, depth:9 | -0.1058 | -0.062 | |
| Adaboost DT | ccp-α:0.0001, depth:1,  n\_estimator:201, LR:0.001 | -0.5788 | **-0.6038** | ccp-α:0.00005, depth:5,  n\_estimator:601, LL:0.003 | -0.0755 | -0.0577 | |
| KNN | k: 100, p: 3 | -0.5238 | -0.6338 | k: 200, p: 1 | -0.0868 | -0.071 | |
| SVC | γ: 0.1, C: 0.1, kernel: rbf | **-0.5154** | -0.6389 | γ : 0.0316,C:1, kernel : rbf | -0.0741 | **-0.0558** | |
| NN | batch\_size: 512, hidden\_layer\_sizes: (32, 32), LR\_init: 0.00215, momentum: 0, Early stopping : True | -0.5177 | -0.6172 | batch\_size: 16, hidden\_layer\_sizes: (64,), LR\_init: 0.03162, momentum:1e-04 Early stopping : True | **-0.0726** | 0.0662 | |

* 1. **Intra-data:**
* **Dataset 1**

While SVC creates the best average estimator with highest CV NLL of -0.5154, the difference between performance of algorithms is not very significant, that is all models performance relatively poor on the data with very low NLL, this points depicts how unbalanced and skewed distributions, along with irrelevant features fail identify a good estimate of target concept.

Additionally, models with similar or best CV accuracies may not equally generalize test data, that is it is possible to overfit CV data. The worst performing model over CV, Adaboost, generalized better than all other additionally over test set, NLL -0.6038, which also indicates the how it decrease possible overfit over DT, which performs the worst over test set.

* **Dataset 2**

All estimators perform relatively well on CV as well as test data, with best being NN with CV NLL -0.0726, and worst DT at CV NLL -0.1058, which indicates much complex feature can could segregate data better. This also indicates, when the data is well balanced, show associated with target function, and creates limited concept space, then any method when tuned can segregate instances well.

Following the observation from dataset 1, when the models have similar training or cv performances, the expected performance on unseen data is more driven by characteristics of data, even though DT had lower CV NLL, it performed better than NN on test. And SVC performed better on test than any other model, while performing comparable to NN on CV, that might indicate support of inherent separate of instances based on features, which NN might not have captured so well.

Additionally the performance of models across test was found better than CV. Given the good overall performance of the models and sufficient training data, this may indicate there could be specific CV sets with high noise. Therefore it would be beneficial to validate the performance of all CV sets.

* 1. **Performance comparison across Data**

From the above analysis the importance of balanced and relevant feature set is visible, therefore it is imperative to analyse the data before training a model. This fact is supported by all the algorithms, which performed better on dataset 2. Additionally, it can be seen that NN and SVC could identify patterns regardless description of data, using higher dimensions or derived features, while giving acceptable performance on unseen test data.

* 1. **Conclusion.**

This Study explores the working of different algorithms on different type of data, highlighting key driving factors of each. A major insight from showcase the importance of description of dataset itself before application of any algorithm, which can limit performance throughout the spectrum of algorithms. Additionally the study highlights the adaptability of SVM and Neural networks to different kind of data distributions and prediction problems.

* + Describe algorithm and parameters explored
  + Result per data 1
    - Loss curve
      * Overfitting underfitting
      * Pref and inductive bias
    - Learning curve
  + Result for data 2
    - Loss curve
      * Overfitting underfitting
      * Pref and inductive bias
    - Learning curve
      * Overfitting underfitting
      * Pref and inductive bias
  + Compare results between data
    - Loss curve
      * Overfitting underfitting
      * Pref and inductive bias
    - Learning curve
      * Overfitting underfitting
      * Pref and inductive bias

**Reference document:**

when can overfit and underfit happen in the above models

Overfitting and underfitting can occur in various machine learning models, including decision trees, AdaBoost decision trees, neural networks, support vector machines (SVMs), and k-nearest neighbors (KNN). The occurrence of these issues depends on factors like model complexity, dataset size, and the choice of hyperparameters. Here's a breakdown of when overfitting and underfitting can happen in each of these models:

1. \*\*Decision Trees\*\*:

- \*\*Overfitting\*\*: Decision trees can easily overfit if they are allowed to become too deep or complex. Overfitting occurs when the tree captures noise and small variations in the training data, resulting in poor generalization to unseen data.

- \*\*Underfitting\*\*: Underfitting may happen if the decision tree is too shallow or simple to capture the underlying patterns in the data. This leads to high training error and poor performance on both the training and validation datasets.

2. \*\*AdaBoost Decision Trees\*\*:

- \*\*Overfitting\*\*: While AdaBoost can reduce overfitting compared to individual decision trees, it is still possible to overfit if the base decision trees are very deep or if the boosting process continues for too many iterations.

- \*\*Underfitting\*\*: Underfitting in AdaBoost is less common, as the ensemble tends to focus on improving performance on misclassified examples. However, if the base learners are too weak, the ensemble may underfit the data.

3. \*\*Neural Networks\*\*:

- \*\*Overfitting\*\*: Neural networks are prone to overfitting when they have a large number of parameters relative to the size of the dataset. This occurs when the network learns to fit noise in the training data, resulting in excellent training performance but poor generalization to new data.

- \*\*Underfitting\*\*: Underfitting can occur if the neural network is too shallow or lacks the capacity to capture complex patterns in the data. In such cases, the model may struggle to fit the training data effectively.

4. \*\*Support Vector Machines (SVM)\*\*:

- \*\*Overfitting\*\*: SVMs can overfit when the kernel used is too complex or when the regularization parameter (C) is set too low. Overfitting leads to a narrow margin and misclassification of training data.

- \*\*Underfitting\*\*: SVMs may underfit when the kernel is too simple or when C is set too high. Underfitting results in a wide margin that does not effectively separate classes.

5. \*\*K-Nearest Neighbors (KNN)\*\*:

- \*\*Overfitting\*\*: KNN is less prone to overfitting because it makes predictions based on local patterns. However, if you set a very small value of k (e.g., k=1), the model may become sensitive to noise in the training data.

- \*\*Underfitting\*\*: KNN can underfit if you choose a very large value of k that leads to overly smooth decision boundaries, making it difficult to capture local patterns.

To mitigate overfitting and underfitting in these models, you can apply techniques such as cross-validation, regularization, early stopping, feature selection, and hyperparameter tuning. The appropriate measures to combat these issues will depend on the specific model and problem you are addressing.

**FROM BARD**

Overfitting and underfitting can happen in any machine learning model, including decision trees, Adaboost decision trees, neural networks, support vector machines, and k-nearest neighbors.

**Overfitting**

Overfitting happens when a model learns the training data too well, to the extent that it is unable to generalize to new data. This can happen when the model is too complex or when the training set is too small.

**Underfitting**

Underfitting happens when a model is unable to learn the training data well enough. This can happen when the model is too simple or when the training set is too small.

Here are some specific examples of when overfitting and underfitting can happen in the above models:

**Decision trees**

* Overfitting: If a decision tree is too deep, it may learn the noise in the training data, which can lead to overfitting.
* Underfitting: If a decision tree is too shallow, it may not be able to learn the complex relationships in the training data, which can lead to underfitting.

**Adaboost decision trees**

* Overfitting: If Adaboost is trained for too many iterations, it may overfit the training data.
* Underfitting: If Adaboost is not trained for enough iterations, it may not be able to learn the complex relationships in the training data, which can lead to underfitting.

**Neural networks**

* Overfitting: Neural networks are very prone to overfitting, especially if they have too many parameters or if the training set is too small.
* Underfitting: If a neural network has too few parameters or if the training set is too small, it may not be able to learn the complex relationships in the training data, which can lead to underfitting.

**Support vector machines (SVMs)**

* Overfitting: If an SVM is trained with a large regularization parameter, it may overfit the training data.
* Underfitting: If an SVM is trained with a small regularization parameter, it may not be able to learn the complex relationships in the training data, which can lead to underfitting.

**K-nearest neighbors (KNN)**

* Overfitting: If KNN is used with a small value of K, it may overfit the training data.
* Underfitting: If KNN is used with a large value of K, it may not be able to learn the complex relationships in the training data, which can lead to underfitting.

**How to avoid overfitting and underfitting**

There are a number of things that can be done to avoid overfitting and underfitting, such as:

* **Use a validation set:** A validation set is a set of data that is not used to train the model, but is used to evaluate the model's performance on unseen data. This can help to identify overfitting.
* **Use regularization:** Regularization is a technique that penalizes the model for complexity. This can help to prevent overfitting.
* **Use early stopping:** Early stopping is a technique that stops the training process early, before the model has a chance to overfit the training data.
* **Use a larger training set:** A larger training set can help to reduce the risk of overfitting and underfitting.

It is important to note that there is no one-size-fits-all solution to avoiding overfitting and underfitting. The best approach will depend on the specific problem and the type of data being used.

what are the major differences in their loss curves and learning curves

The loss curves and learning curves of machine learning models provide valuable insights into how models are learning from data during the training process. While the specific shapes and patterns of these curves can vary depending on the model, dataset, and hyperparameters, I can provide a general overview of the major differences you might observe in the loss curves and learning curves of different types of models:

1. \*\*Loss Curves\*\*:

- \*\*Decision Trees\*\*: Decision trees typically do not have loss curves in the traditional sense because they are not trained iteratively using gradient-based optimization. Instead, they construct the tree structure by recursively splitting data based on information gain or Gini impurity, leading to discrete, step-like behavior.

- \*\*AdaBoost Decision Trees\*\*: AdaBoost loss curves often show a decreasing trend over iterations. As the algorithm iteratively focuses on misclassified examples, it reduces the training error, resulting in a declining loss curve.

- \*\*Neural Networks\*\*: Neural networks exhibit smooth, continuously decreasing loss curves during training. You may observe fluctuations or plateaus in the early stages of training, but as training progresses, the loss typically decreases. Overfitting can manifest as an increasing loss on a validation set while the training loss continues to decrease.

- \*\*Support Vector Machines\*\*: SVMs aim to minimize the hinge loss, which often leads to a convex optimization problem. The loss curve for SVMs is generally smooth and continuously decreasing as the optimization algorithm converges to a solution.

- \*\*K-Nearest Neighbors\*\*: KNN does not have a traditional loss curve since it doesn't have a loss function that is optimized during training. Instead, it computes distances and uses majority voting during prediction.

2. \*\*Learning Curves\*\*:

- \*\*Decision Trees\*\*: Decision trees tend to have low bias and high variance. Learning curves for decision trees may show that as you increase the sample size, the training and validation error initially decrease. However, the validation error may stabilize or increase again as the tree becomes more complex and overfits the data.

- \*\*AdaBoost Decision Trees\*\*: Learning curves for AdaBoost often show a reduction in training and validation error as the algorithm iteratively focuses on misclassified examples. AdaBoost can effectively reduce bias and variance.

- \*\*Neural Networks\*\*: Neural networks may show learning curves with a decreasing training error and a possibly fluctuating or increasing validation error, indicating overfitting. Increasing the dataset size or using regularization techniques can help alleviate overfitting.

- \*\*Support Vector Machines\*\*: SVMs are robust to overfitting, and learning curves may show that both training and validation errors decrease as you increase the dataset size. They are effective for high-dimensional data.

- \*\*K-Nearest Neighbors\*\*: KNN's learning curves may demonstrate that training and validation errors decrease as you increase the sample size, but they often exhibit high computational complexity due to the need to compute distances for each prediction.

It's important to note that the specific shapes of these curves can vary based on factors such as dataset size, data complexity, model architecture, and hyperparameter settings. The goal when analyzing these curves is to strike a balance between bias and variance, and to identify points where further data collection, model selection, or hyperparameter tuning can improve model performance.

what are the preference and inductive biases in 1. Decision trees. 2. Adaboost Decision trees 3. Neural networks 4. Support vector machines 5. K nearest neighbours, also tell what are the major performance difference between these models with example

Inductive biases and preferences refer to the inherent assumptions and tendencies of machine learning models when making predictions or learning from data. Each machine learning algorithm has its own set of biases and preferences, which influence how they handle data and make decisions. Here are the inductive biases and preferences for the specified models:

1. \*\*Decision Trees\*\*:

- \*\*Bias Towards Axis-Aligned Splits\*\*: Decision trees tend to make splits along feature axes, meaning they create orthogonal decision boundaries.

- \*\*Preference for Simple Models\*\*: Decision trees aim to create simple, interpretable models by selecting features and thresholds that lead to the most significant information gain or Gini impurity reduction.

- \*\*Top-Down Recursive Partitioning\*\*: They follow a top-down approach, recursively splitting the data based on the most informative features.

2. \*\*AdaBoost Decision Trees\*\*:

- \*\*Bias Towards Correcting Errors\*\*: AdaBoost prefers to focus on misclassified examples from previous weak learners. It assigns higher weights to those examples, effectively reducing the model's bias towards previously misclassified data points.

- \*\*Preference for Combining Weak Learners\*\*: AdaBoost combines multiple decision trees (typically weak learners) to form a strong ensemble, leveraging the strengths of each while mitigating their weaknesses.

3. \*\*Neural Networks\*\*:

- \*\*Bias Towards Nonlinear Relationships\*\*: Neural networks are highly flexible and can model complex, nonlinear relationships between features and target variables.

- \*\*Preference for Learning Hierarchical Features\*\*: They learn features at different levels of abstraction through hidden layers, enabling them to capture hierarchical patterns in data.

- \*\*Capacity to Overfit\*\*: Neural networks have a high capacity to overfit the data if not properly regularized, leading to a preference for large amounts of data and regularization techniques.

4. \*\*Support Vector Machines (SVM)\*\*:

- \*\*Bias Towards Large Margin\*\*: SVMs aim to find a hyperplane with the maximum margin between classes. They prefer solutions that maintain a large margin between data points of different classes.

- \*\*Preference for Linear Separability\*\*: SVMs work well when data is linearly separable, and they may struggle with highly nonlinear data without the use of kernel tricks.

- \*\*Sparsity in Support Vectors\*\*: In many cases, only a subset of data points (support vectors) significantly affects the model's decision boundaries.

5. \*\*K-Nearest Neighbors (KNN)\*\*:

- \*\*Bias Towards Local Patterns\*\*: KNN makes predictions based on the majority class among its k-nearest neighbors. It assumes that similar instances are likely to belong to the same class.

- \*\*Preference for Distance Metrics\*\*: The choice of distance metric (e.g., Euclidean, Manhattan) can affect KNN's performance and bias towards different types of patterns.

- \*\*Sensitivity to the Value of k\*\*: The number of neighbors (k) is a hyperparameter in KNN that affects model bias. Smaller values of k may lead to a more flexible (less biased) model, while larger values of k may introduce more bias.

Regarding the major performance differences between these models, it depends on the nature of the data and the specific problem. Here are some general considerations:

- \*\*Decision Trees vs. AdaBoost Decision Trees\*\*: AdaBoost often outperforms a single decision tree in terms of predictive accuracy by combining multiple trees. It reduces overfitting and can be more robust.

- \*\*Decision Trees vs. Neural Networks\*\*: Neural networks are highly expressive and capable of modeling complex relationships. They tend to outperform decision trees on tasks involving image recognition, natural language processing, and other complex data types.

- \*\*Support Vector Machines vs. KNN\*\*: SVMs are suitable for high-dimensional data and cases where there is a clear separation between classes. KNN works well when data is not linearly separable and can capture local patterns.

- \*\*Neural Networks vs. KNN\*\*: Neural networks can handle both structured and unstructured data, while KNN is primarily used for structured data. Neural networks are more flexible but require larger amounts of data and longer training times.

Performance differences between models are best assessed empirically on specific datasets and tasks. It's important to consider factors like data size, data complexity, computational resources, and interpretability when choosing a model. There is no one-size-fits-all answer, and model selection should be based on the problem at hand.

**FROM BARD:**

Decision trees

Preference bias: Decision trees prefer features that have a high variance and are less sensitive to outliers.

Inductive bias: Decision trees assume that the target variable is a function of the features in a piecewise linear fashion.

Loss curve: The loss curve for decision trees is typically non-convex, meaning that it may have multiple local minima.

Learning curve: The learning curve for decision trees is typically steep at first, as the tree learns the most important features. As the tree grows larger, the learning curve flattens out.

Adaboost decision trees

Preference bias: Adaboost decision trees inherit the preference bias of decision trees.

Inductive bias: Adaboost decision trees assume that the target variable can be predicted by a weighted combination of simple decision trees.

Loss curve: The loss curve for Adaboost decision trees is typically convex, meaning that it has a single global minimum. This makes it easier to train Adaboost decision trees than decision trees.

Learning curve: The learning curve for Adaboost decision trees is typically similar to the learning curve for decision trees, but it may be smoother due to the convexity of the loss curve.

Neural networks

Preference bias: Neural networks do not have a strong preference bias for any particular type of feature.

Inductive bias: Neural networks assume that the target variable is a function of the features in a non-linear fashion.

Loss curve: The loss curve for neural networks is typically non-convex, meaning that it may have multiple local minima.

Learning curve: The learning curve for neural networks is typically slow at first, as the network learns the complex relationships between the features and the target variable. As the network trains for longer, the learning curve steepens and eventually flattens out.

Support vector machines (SVMs)

Preference bias: SVMs prefer features that have a high margin between the positive and negative classes.

Inductive bias: SVMs assume that the target variable can be separated from the features by a hyperplane with a large margin.

Loss curve: The loss curve for SVMs is typically convex, meaning that it has a single global minimum. This makes it easier to train SVMs than neural networks.

Learning curve: The learning curve for SVMs is typically similar to the learning curve for decision trees, but it may be smoother due to the convexity of the loss curve.

K-nearest neighbors (KNN)

Preference bias: KNN does not have a strong preference bias for any particular type of feature.

Inductive bias: KNN assumes that the target variable of a new data point is equal to the target variable of the K most similar data points in the training set.

Loss curve: The loss curve for KNN is typically non-convex, meaning that it may have multiple local minima.

Learning curve: The learning curve for KNN is typically flat, as the performance of KNN does not improve significantly as the training set grows larger.

Major differences in loss curves and learning curves

The main difference between the loss curves of decision trees, Adaboost decision trees, neural networks, and SVMs is that the loss curves for Adaboost decision trees and SVMs are convex, while the loss curves for decision trees and neural networks are non-convex. This means that Adaboost decision trees and SVMs are easier to train than decision trees and neural networks, as they are less likely to get stuck in local minima.

The main difference between the learning curves of decision trees, Adaboost decision trees, neural networks, and SVMs is that the learning curve for KNN is flat, while the learning curves for the other algorithms are typically steep at first and then flatten out. This is because KNN does not learn from the training data in the same way that the other algorithms do.

Overall

Different machine learning algorithms have different preference biases and inductive biases. These biases can affect the performance of the algorithms on different types of data. It is important to choose the right algorithm for the specific problem you are trying to solve.

Version 2

**Decision trees**

Decision trees are a type of machine learning model that uses a tree-like structure to make predictions. Decision trees prefer to learn relationships between features that can be represented by simple rules. For example, a decision tree might learn the rule "If the temperature is above 80 degrees Fahrenheit and the humidity is above 50%, then it is likely to rain."

**Adaboost decision trees**

Adaboost decision trees are an ensemble learning method that uses multiple decision trees to make predictions. Adaboost decision trees prefer to learn relationships between features that are difficult for individual decision trees to learn. For example, an Adaboost decision tree might learn to predict whether or not a customer will churn by combining the predictions of multiple decision trees that are trained on different subsets of the data.

**Neural networks**

Neural networks are a type of machine learning model that is inspired by the structure of the human brain. Neural networks prefer to learn complex relationships between features that are difficult to represent with simple rules. For example, a neural network might learn to predict the price of a house by combining the features of the house, such as its location, size, and number of bedrooms, in a complex way.

**Support vector machines**

Support vector machines are a type of machine learning model that can be used for both classification and regression tasks. Support vector machines prefer to learn relationships between features that can be represented by linear hyperplanes. For example, a support vector machine might learn to predict whether or not a customer will churn by finding a linear hyperplane that separates the customers who churned from the customers who did not churn.

**K nearest neighbors**

K nearest neighbors is a type of machine learning model that makes predictions based on the most similar data points in the training set. K nearest neighbors does not have any specific preferences or inductive biases. Instead, it learns from the data that it is trained on.

**Differences in loss curves and learning curves**

The loss curve of a machine learning model shows how the loss of the model changes as it is trained. The learning curve of a machine learning model shows how the accuracy of the model changes as it is trained.

The loss curves and learning curves of decision trees, Adaboost decision trees, neural networks, support vector machines, and k nearest neighbors can vary depending on the specific problem and dataset. However, there are some general trends that can be observed.

**Decision trees**

Decision trees typically have a high initial loss, but they can learn quickly and achieve good accuracy with a relatively small amount of training data.

**Adaboost decision trees**

Adaboost decision trees typically have a lower initial loss than decision trees, and they can achieve even better accuracy with more training data.

**Neural networks**

Neural networks typically have a high initial loss, and they can take a long time to train. However, neural networks can achieve very good accuracy on complex problems, given enough training data.

**Support vector machines**

Support vector machines typically have a lower initial loss than neural networks, and they can train faster. However, support vector machines may not be able to achieve as good accuracy as neural networks on complex problems.

**K nearest neighbors**

K nearest neighbors does not typically have a high initial loss, but it can be slow to train on large datasets.

Overall, the best machine learning algorithm for a particular problem depends on the specific problem and dataset. However, the preferences and inductive biases of different machine learning algorithms can provide a good starting point for choosing an algorithm.

Here are some additional tips for choosing a machine learning algorithm:

* Consider the complexity of the problem. If the problem is complex, then you may need to use a more complex algorithm, such as a neural network.
* Consider the size of the dataset. If the dataset is small, then you may need to use an algorithm that is able to learn from a small amount of data, such as a decision tree.
* Consider the computational resources available to you. Some algorithms, such as neural networks, can be computationally expensive to train.
* Experiment with different algorithms. The best way to find the best algorithm for a particular problem is to experiment with different algorithms and see which one performs best.

Version 3

**Decision trees**

**Preference:** Decision trees prefer simple rules and avoid overfitting. **Inductive bias:** Decision trees assume that the data can be represented by a set of simple rules.

**Adaboost decision trees**

**Preference:** Adaboost decision trees prefer a diverse set of weak learners. **Inductive bias:** Adaboost decision trees assume that the data can be represented by a combination of weak learners.

**Neural networks**

**Preference:** Neural networks prefer complex relationships between features. **Inductive bias:** Neural networks assume that the data can be represented by a non-linear function.

**Support vector machines (SVMs)**

**Preference:** SVMs prefer a hyperplane that maximizes the margin between the two classes. **Inductive bias:** SVMs assume that the data can be separated by a hyperplane.

**K-nearest neighbors (KNN)**

**Preference:** KNN prefers to learn from the most similar data points. **Inductive bias:** KNN assumes that the data is locally smooth.

**Loss curves and learning curves**

The loss curve shows how the loss of the model changes as it is trained. The learning curve shows how the accuracy of the model changes as it is trained.

**Decision trees**

Decision trees typically have a smooth loss curve and learning curve. This is because decision trees are relatively simple models and are not prone to overfitting.

**Adaboost decision trees**

Adaboost decision trees typically have a bumpy loss curve and learning curve. This is because Adaboost decision trees are a combination of weak learners, which are not individually very accurate. However, as the model is trained, the weak learners are combined to create a more accurate model.

**Neural networks**

Neural networks can have very different loss curves and learning curves, depending on the architecture of the network and the complexity of the data. However, neural networks are often prone to overfitting, which can lead to a bumpy loss curve and learning curve.

**Support vector machines (SVMs)**

SVMs typically have a smooth loss curve and learning curve. This is because SVMs are relatively simple models and are not prone to overfitting. However, SVMs can be slow to train, especially on large datasets.

**K-nearest neighbors (KNN)**

KNN typically has a smooth loss curve and learning curve. This is because KNN is a non-parametric model, which means that it does not make any assumptions about the distribution of the data. However, KNN can be slow to make predictions, especially on large datasets.

**Conclusion**

The preference and inductive biases of a machine learning algorithm determine how the algorithm learns from data and makes predictions. Different algorithms have different preferences and inductive biases, which makes them suitable for different types of data and problems.

The loss curve and learning curve of a machine learning algorithm show how the algorithm performs as it is trained. The loss curve shows how the loss of the model changes as it is trained, while the learning curve shows how the accuracy of the model changes as it is trained.

The loss curve and learning curve of a machine learning algorithm can be used to assess the performance of the algorithm and to identify potential problems, such as overfitting.