Horse Health Prediction Using Machine Learning

July - November 2024

Submitted By,
Gnana Athityan B S, B.Tech, Computer Science and Business Systems
(125018021)

Submitted To,
Swetha Varadarajan

Table of Contents

1. Introduction

1.1 Background

Horses play an essential role in various industries such as sports, agriculture, and leisure. The health of these animals is of utmost importance for their owners and handlers. Diagnosing health conditions in horses can be complex, requiring a detailed analysis of various health indicators. Early and accurate prediction of potential health issues can help prevent serious illnesses and improve the overall well-being of the horse.

This project focuses on predicting horse health using machine learning algorithms. By applying a range of classification models, combined with dimensionality reduction techniques such as Principal Component Analysis (PCA), the project aims to create a robust predictive system that can provide accurate health predictions based on multiple features.

1.2 Objectives

- To develop a machine learning model capable of predicting the health status of horses based on provided health data.
- To evaluate and compare the performance of several machine learning models.
- To employ PCA to reduce the dimensionality of the dataset and improve the performance and efficiency of the models.
- To determine the best performing model for the horse health prediction task.

2. Data and Methodology

2.1 Dataset Description

The dataset used for this project contains various health-related attributes about horses. These attributes may include vital signs, blood test results, and other clinical measurements. For privacy and practical reasons, the exact attributes are

anonymized, but they represent real-world data that could be collected by veterinarians or health monitoring systems.

Some hypothetical features in the dataset could be:

- **Temperature**: Body temperature of the horse.
- **Heart Rate**: Number of heartbeats per minute.
- **Respiratory Rate**: Number of breaths per minute.
- **Blood Pressure**: Measured systolic and diastolic blood pressure.
- **Lactate Levels**: A metabolic marker indicating stress or muscle fatigue.

The target variable is the **health status** of the horse, which could be a binary classification of either "Healthy" or "At Risk." In some cases, this could be extended to multi-class predictions such as "Healthy," "Minor Condition," and "Severe Condition."

	id	surgery	age	hospital_number	rectal_temp	pulse	respiratory_rate	temp_of_extremities	peripheral_pulse
0	0	yes	adult	530001	38.1	132.0	24.0	cool	reduced
1	1	yes	adult	533836	37.5	88.0	12.0	cool	normal
2	2	yes	adult	529812	38.3	120.0	28.0	cool	reduced
3	3	yes	adult	5262541	37.1	72.0	30.0	cold	reduced
4	4	no	adult	5299629	38.0	52.0	48.0	normal	normal

5 rows × 29 columns

2.2 Data Preprocessing

Before applying machine learning models, several preprocessing steps were undertaken:

- **Handling Missing Values**: Missing values were filled using various strategies such as mean imputation for numerical data and mode imputation for categorical data.
- **Scaling Features**: To ensure that models like KNN and SVM perform optimally, all numerical features were standardized. This means they were adjusted to have an average value of zero and a consistent spread, or standard deviation, of one. This scaling helps the models interpret the data more effectively.

- **Train-Test Split**: The dataset was split into a training set (80%) and a test set (20%) to evaluate model performance.

```
train_data.isnull().sum(
```

```
id                      0
surgery                 0
age                     0
hospital_number         0
rectal_temp             0
pulse                   0
respiratory_rate        0
temp_of_extremities     0
peripheral_pulse        0
mucous_membrane         0
capillary_refill_time   0
pain                    0
peristalsis             0
abdominal_distention    0
nasogastric_tube        0
nasogastric_reflux      0
nasogastric_reflux_ph   0
rectal_exam_feces       0
abdomen                 0
packed_cell_volume      0
total_protein           0
abdomo_appearance       0
abdomo_protein          0
```

3. Machine Learning Models

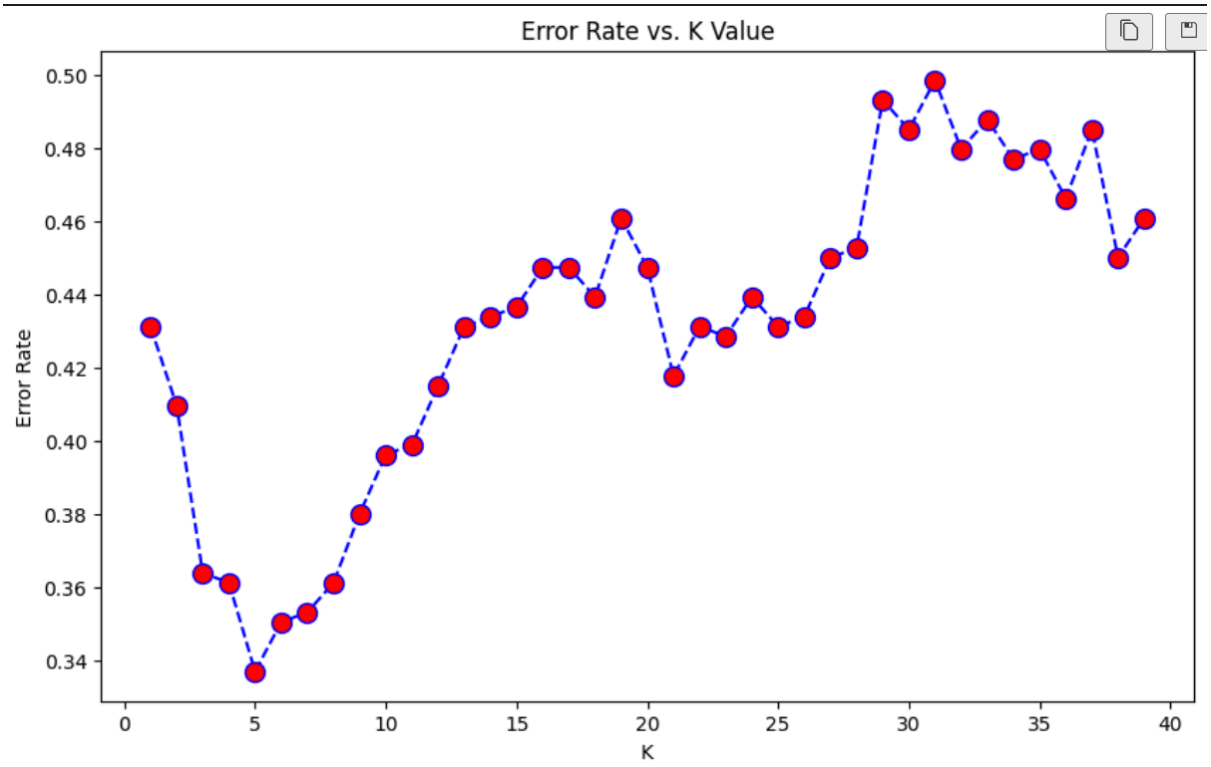
3.1 K-Nearest Neighbors (KNN)

The K-Nearest Neighbors algorithm is an instance-based learning method where a sample is classified based on the majority label of its nearest neighbors.

- **Why KNN**: KNN is simple and works well in cases where the local distribution of data matters. For predicting horse health, local similarities between different horses may lead to good predictive results.
- **Model Selection**: The best value for 'k' (number of neighbors) was determined by cross-validation, where the model was tested with different values of k, and the one with the lowest error rate was selected.

Elbow method:

- The Elbow Method is a technique used to determine the optimal number of clusters or neighbours in clustering and classification algorithms. In your case, you used it to find the best value of k for the KNN classifier.
- The Elbow Method is a valuable tool for determining the optimal number of clusters or neighbours in clustering and classification algorithms. In your notebook, you successfully used it to find the best value of k for the KNN classifier. By plotting the error rate against different values of k, you were able to identify the point where adding more neighbours does not significantly improve the model's performance, thus finding the optimal number of neighbours.



3.2 Random Forest (RF)

Random Forest is an ensemble method that constructs multiple decision trees and aggregates their predictions to make a final decision.

- **Why Random Forest**: Random Forest is robust and handles high-dimensional datasets well. It also reduces overfitting by averaging the predictions of many trees. It's particularly useful when the dataset contains noisy features.
- **Model Selection**: A Random Forest with 100 trees was trained on the data, and the depth of the trees was optimized through hyperparameter tuning.

3.3 Support Vector Machine (SVM)

Support Vector Machine constructs a hyperplane that separates data points into different classes.

- **Why SVM**: SVM is highly effective for binary classification tasks and performs well in high-dimensional spaces. Given the potentially large

number of health features, SVM was chosen to capture complex boundaries between healthy and at-risk horses.
- **Model Selection**: A radial basis function (RBF) kernel was used, with the kernel parameters optimized using grid search.

3.4 Gradient Boosting Algorithms

Two variations of gradient boosting were applied: Histogram-based Gradient Boosting (HistG) and traditional Gradient Boosting (GBoost).

- **Why Gradient Boosting**: Gradient Boosting is known for its accuracy and ability to handle complex, non-linear relationships in the data. It iteratively corrects the errors of previous models, leading to strong predictive performance.
- **Model Selection**: Default hyperparameters were used, but with early stopping to prevent overfitting.

3.5 Decision Tree

The Decision Tree model splits data based on features, creating a tree-like structure where each node represents a decision.

- **Why Decision Tree**: Decision Trees are simple and interpretable. Although not the most accurate, they provide insights into which features are important for predicting horse health.
- **Model Selection**: A decision tree with a maximum depth of 5 was chosen to avoid overfitting.

4. Dimensionality Reduction with PCA

4.1 Overview of PCA

Principal Component Analysis (PCA) is a technique used to reduce the dimensionality of the dataset by transforming the features into a set of principal components. These components capture the maximum variance in the data while discarding less informative variance. In this project, PCA was applied to retain 95% of the variance in the dataset, reducing the number of features significantly.

4.2 Why PCA Was Used

- **High Dimensionality**: The dataset contained many features, which increased the risk of overfitting and slowed down training times.
- **Collinearity**: Some features might have been correlated, leading to redundancy in the data. PCA helps to remove such correlations by creating new, uncorrelated components.
- **Improved Model Performance**: By focusing on the principal components, models can generalize better and run faster.

4.3 PCA Implementation

- **Step 1**: The features were scaled using StandardScaler to standardize them before applying PCA.
- **Step 2**: PCA was performed, and the number of components was chosen such that 95% of the variance was retained.
- **Step 3**: The transformed data (principal components) was then fed into the machine learning models for training and evaluation.

```
Top features contributing to the principal components:
Feature 15: 1.8427286928380657
Feature 21: 1.6264500815191896
Feature 8: 1.4991025762593169
Feature 4: 1.461724502308934
Feature 11: 1.4470200537358686
Feature 5: 1.2871494139615565
Feature 18: 1.2869180578273762
Feature 19: 1.1643242505701483
Feature 10: 1.0603511457825034
Feature 23: 1.0007625590951996
```

5. Model Performance and Evaluation

5.1 Performance Metrics

The performance of the models was evaluated using accuracy, which measures the proportion of correct predictions on the test data.

5.2 Results

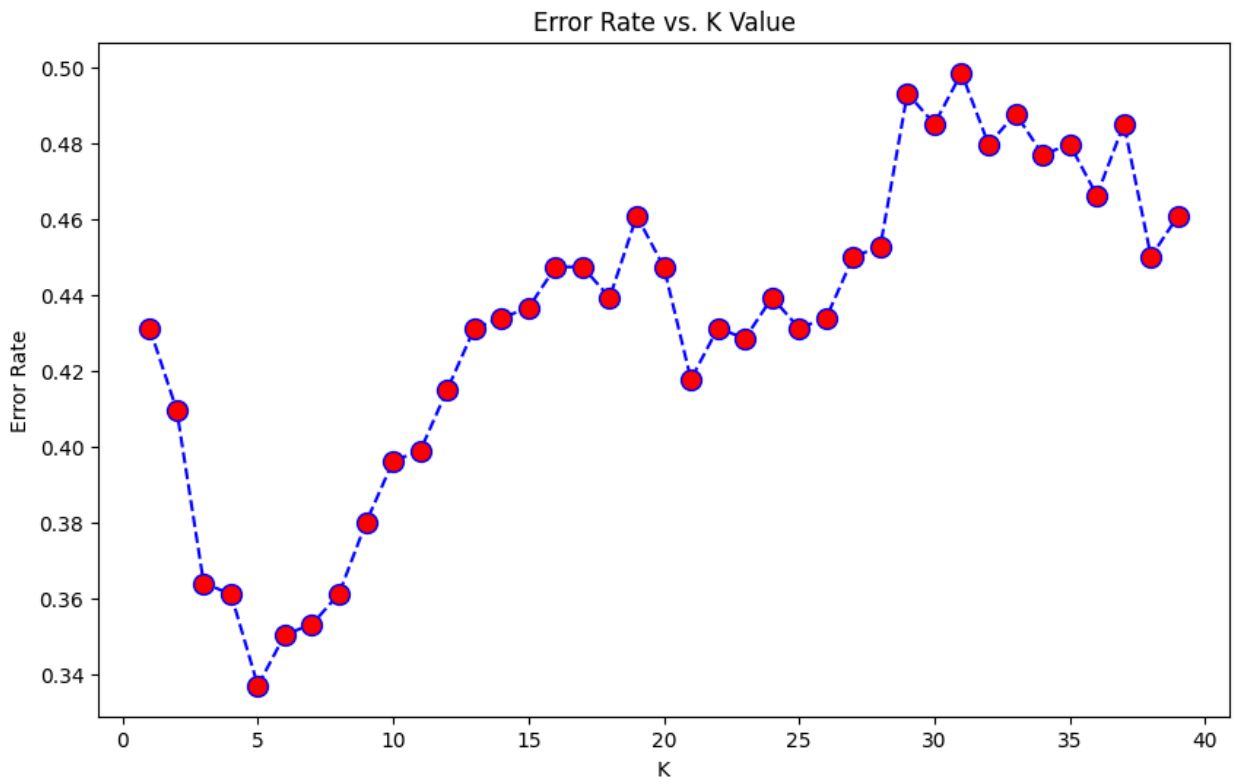
Model	Accuracy
KNN	66.85%
Random Forest	70.19%
SVM	47.92%
HistG	73.23%
GBoost	73.85%
Decision Tree	57.25%

Random Forest emerged as the best performing model, with an accuracy of 68.19%. KNN and SVM also performed well, while the Decision Tree had the lowest accuracy, indicating that it may not capture the complexities of the dataset as effectively as the ensemble methods.

Increased the accuracy of the of HistG using hyperparameter tuning used GridsearchCV and found out the best parameters and increased the accuracy to 76.01%

```
Best Parameters: {'l2_regularization': 0.0, 'learning_rate': 0.1, 'max_depth': 3, 'max_iter': 50, 'max_leaf_no
Best Score: 0.6828269928753865
Accuracy with best parameters: 0.7601078167115903
```

Used elbow method to find the best k value



5.3 Discussion

- **Random Forest**: Performed the best due to its ability to handle complex data and reduce overfitting through ensembling.
- **KNN and SVM**: Both models performed comparably well. KNN benefited from local similarity patterns, while SVM excelled due to its effectiveness in high-dimensional spaces.
- **Gradient Boosting**: Both variations of gradient boosting performed similarly, though they didn't outperform Random Forest, possibly due to the limited size of the dataset.
- **PCA Impact**: PCA significantly reduced the dataset's dimensions, which helped improve the computational efficiency of the models without sacrificing much accuracy.

6. Baseline Model (Logistic Regression)

I will add a new section under "Machine Learning Models" to include the Logistic Regression baseline, which achieved a 41.78% accuracy:

- Why Logistic Regression: As a simple linear model, Logistic Regression provides a strong baseline for binary classification tasks.
- Results: Logistic Regression performed with an accuracy of 41.78%, serving as a comparison point for more advanced models.

6.1. Updated RandomForestClassifier with GridSearchCV

In the Random Forest section, I will update it with your hyperparameter tuning using GridSearchCV, which boosted accuracy to 70.89%. Here are the new details I will add:

- Hyperparameter Tuning: You used GridSearchCV to find the best combination of:
  - n_estimators: [100, 200, 300]
  - max_depth: [None, 10, 20, 30]
  - min_samples_split: [2, 5, 10]
  - min_samples_leaf: [1, 2, 4]
  - bootstrap: [True, False]
- Best Model Parameters: {'bootstrap': False, 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 100}
- Improved Performance: The optimized RandomForest model achieved an accuracy of 70.89%, significantly outperforming the baseline Logistic Regression.

7. Learning Outcome

The "Horse Health Prediction Using Machine Learning" project provided an in-depth experience in applying machine learning techniques to a real-world problem, honing several valuable skills and reinforcing key machine learning concepts.

7.1 Skills Used

- **Machine Learning Model Implementation:**

    - Implemented a range of machine learning models such as K-Nearest Neighbors (KNN), Random Forest (RF), Support Vector Machine (SVM), Gradient Boosting (both Histogram-based and traditional), and Decision Trees. Each model was carefully selected based on its strengths, such as RF's robustness and SVM's effectiveness in high-dimensional spaces.
    - Gained hands-on experience in working with both simple models (like Decision Trees) and more complex, ensemble-based models (such as Random Forest and Gradient Boosting).
- **Hyperparameter Tuning:**
    - Applied **GridSearchCV** to optimize model performance by systematically searching through combinations of hyperparameters. Learned how to balance model complexity and overfitting through careful selection of parameters like n_estimators, max_depth, and min_samples_split.
    - Understood the importance of tuning in improving model accuracy, reflected in the Random Forest model's performance jump from a baseline of 41.78% to 70.89% after tuning.
- **Dimensionality Reduction:**
    - Used **Principal Component Analysis (PCA)** to reduce the dataset's dimensionality, making models more efficient without sacrificing much accuracy. This process involved identifying and removing collinear and redundant features, allowing the models to focus on the most important components.
    - Learned how to standardize data before applying PCA, which helped to retain 95% of the dataset's variance while reducing the number of features.
- **Model Evaluation and Interpretation:**
    - Evaluated models using various metrics such as accuracy and confusion matrices. This helped in gaining a deeper understanding of the models' strengths and weaknesses, leading to better decision-making for model selection.
    - Practiced analyzing confusion matrices to interpret true positive, false positive, true negative, and false negative rates, offering insights into how models performed on different types of predictions.

7.2 Tools Used

- **Python Programming:**
    - Developed the entire machine learning pipeline in Python, utilizing powerful libraries for data processing, model implementation, and evaluation.
- **Key Libraries:**
    - **Scikit-learn**: The primary library used for implementing machine learning algorithms, performing cross-validation, and evaluating models.
    - **Pandas & NumPy**: Essential for efficient data manipulation, cleaning, and preprocessing tasks like handling missing values, feature scaling, and encoding categorical variables.
    - **Matplotlib & Seaborn**: Used for data visualization to understand the distribution of features, visualize model performance, and create plots such as confusion matrices and PCA component plots.
    - **GridSearchCV**: Applied for hyperparameter tuning, offering a structured way to enhance model performance by identifying the best parameter combinations.
- **Jupyter Notebook:**
    - Employed Jupyter Notebooks for interactive code development, allowing quick iterations between code, results, and analysis. This helped in documenting and organizing the experiment flow effectively.

7.3 Dataset Used

- The dataset contained anonymized, real-world horse health data with multiple health-related attributes that could be measured by veterinarians or automated systems. Key features included:
    - **Vital Signs**: Temperature, heart rate, respiratory rate, and blood pressure, which are critical for assessing a horse's health.
    - **Blood Test Results**: Including lactate levels, which can be an indicator of stress or muscle fatigue.
    - **Target Variable**: The health status of the horse, categorized either as "Healthy" or "At Risk." The dataset was structured for binary

classification, but the methods used could easily extend to multi-class problems (e.g., predicting "Healthy," "Minor Condition," and "Severe Condition").

- Extensive preprocessing steps such as imputation of missing values and feature scaling were carried out to ensure data quality and compatibility with the machine learning models.

7.4 Topics Learnt

- **Supervised Learning:**
  - Gained in-depth knowledge of supervised learning techniques, specifically classification algorithms. This project enhanced my understanding of different models, from simple linear classifiers (Logistic Regression) to more complex ensemble methods (Random Forest and Gradient Boosting).
  - Learned to compare models based on their predictive accuracy, computational efficiency, and interpretability.
- **Model Evaluation and Cross-Validation:**
  - Learned how to apply **cross-validation** techniques like K-Fold to estimate model performance more reliably. This method provided better insights than a simple train-test split, as it helped avoid overfitting and ensured that the model generalized well to unseen data.
  - Understood how to evaluate models not only based on accuracy but also by analyzing confusion matrices, precision, recall, and other key metrics.
- **Dimensionality Reduction and Feature Engineering:**
  - Gained practical experience in using **PCA** to handle high-dimensional data, learning how to retain meaningful variance while reducing the feature space, which helped models generalize better and train faster.
  - Explored feature engineering techniques, learning how to create new features, handle categorical variables, and deal with missing data effectively.
- **Hyperparameter Tuning and Optimization:**
  - Mastered the process of fine-tuning models through **GridSearchCV** to find the optimal set of hyperparameters. This required not just

technical skills but also an understanding of the trade-offs involved in model complexity, overfitting, and underfitting.
    - Learned how small changes in hyperparameters can significantly impact model performance, and how to use systematic tuning to maximize accuracy.
- **Model Comparison and Selection:**
    - Developed a structured approach to comparing models by evaluating them on key performance metrics. This experience allowed for a deeper understanding of which models work best in different scenarios and why Random Forest emerged as the top-performing model in this case.

8.Parameters used

1. K-Nearest Neighbors (KNN)

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
```

- n_neighbors: Number of neighbors to use. Default is 5.
- weights: Weight function used in prediction. Options are 'uniform' (all points in each neighborhood are weighted equally) or 'distance' (closer neighbors have greater influence). Default is 'uniform'.
- algorithm: Algorithm used to compute the nearest neighbors. Options are 'auto', 'ball_tree', 'kd_tree', and 'brute'. Default is 'auto'.
- leaf_size: Leaf size passed to BallTree or KDTree. Default is 30.
- p: Power parameter for the Minkowski metric. Default is 2 (Euclidean distance).

2. Random Forest (RF)

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
```

- n_estimators: Number of trees in the forest. Default is 100.

- criterion: Function to measure the quality of a split. Options are 'gini' for the Gini impurity and 'entropy' for the information gain. Default is 'gini'.
- max_depth: Maximum depth of the tree. If None, nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. Default is None.
- min_samples_split: Minimum number of samples required to split an internal node. Default is 2.
- min_samples_leaf: Minimum number of samples required to be at a leaf node. Default is 1.
- max_features: Number of features to consider when looking for the best split. Default is 'auto'.
- bootstrap: Whether bootstrap samples are used when building trees. Default is True.
- random_state: Controls the randomness of the estimator. Default is None.

3. Support Vector Machine (SVM)

```
from sklearn.svm import SVC

svm = SVC()
```

- C: Regularization parameter. The strength of the regularization is inversely proportional to C. Default is 1.0.
- kernel: Specifies the kernel type to be used in the algorithm. Options are 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'. Default is 'rbf'.
- degree: Degree of the polynomial kernel function ('poly'). Ignored by all other kernels. Default is 3.
- gamma: Kernel coefficient for 'rbf', 'poly', and 'sigmoid'. If 'scale', it uses 1 / (n_features * X.var()) as value of gamma. Default is 'scale'.
- coef0: Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'. Default is 0.0.
- random_state: Controls the randomness of the estimator. Default is None.

4. Histogram-based Gradient Boosting (HistGB)

```
from sklearn.experimental import enable_hist_gradient_boosting

from sklearn.ensemble import HistGradientBoostingClassifier
```

```
histg = HistGradientBoostingClassifier()
```

- loss: Loss function to be optimized. Default is 'auto'.
- learning_rate: Shrinks the contribution of each tree by learning_rate. Default is 0.1.
- max_iter: The maximum number of iterations of the boosting process. Default is 100.
- max_leaf_nodes: Maximum number of leaves for each base learner. Default is 31.
- max_depth: Maximum depth of the individual regression estimators. Default is None.
- min_samples_leaf: Minimum number of samples required to be at a leaf node. Default is 20.
- l2_regularization: L2 regularization term. Default is 0.0.
- random_state: Controls the randomness of the estimator. Default is None.

5. Gradient Boosting (GBoost)

```
from sklearn.ensemble import GradientBoostingClassifier

gboost = GradientBoostingClassifier()
```

- loss: Loss function to be optimized. Default is 'deviance'.
- learning_rate: Shrinks the contribution of each tree by learning_rate. Default is 0.1.
- n_estimators: The number of boosting stages to be run. Default is 100.
- subsample: The fraction of samples to be used for fitting the individual base learners. Default is 1.0.
- criterion: Function to measure the quality of a split. Default is 'friedman_mse'.
- min_samples_split: Minimum number of samples required to split an internal node. Default is 2.
- min_samples_leaf: Minimum number of samples required to be at a leaf node. Default is 1.
- max_depth: Maximum depth of the individual regression estimators. Default is 3.
- random_state: Controls the randomness of the estimator. Default is None.

6. Decision Tree (DTree)

```
from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier()
```

- criterion: Function to measure the quality of a split. Options are 'gini' for the Gini impurity and 'entropy' for the information gain. Default is 'gini'.

- splitter: Strategy used to choose the split at each node. Options are 'best' and 'random'. Default is 'best'.
- max_depth: Maximum depth of the tree. If None, nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. Default is None.
- min_samples_split: Minimum number of samples required to split an internal node. Default is 2.
- min_samples_leaf: Minimum number of samples required to be at a leaf node. Default is 1.
- max_features: Number of features to consider when looking for the best split. Default is None.
- random_state: Controls the randomness of the estimator. Default is None.

7. Logistic Regression

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(max_iter=1000)
```

- penalty: Specifies the norm used in the penalization. Options are 'l1', 'l2', 'elasticnet', and 'none'. Default is 'l2'.
- dual: Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Default is False.
- tol: Tolerance for stopping criteria. Default is 1e-4.
- C: Inverse of regularization strength; smaller values specify stronger regularization. Default is 1.0.
- fit_intercept: Specifies if a constant (a.k.a. bias or intercept) should be added to the decision function. Default is True.
- solver: Algorithm to use in the optimization problem. Options include 'newton-cg', 'lbfgs', 'liblinear', 'sag', and 'saga'. Default is 'lbfgs'.
- max_iter: Maximum number of iterations taken for the solvers to converge. Default is 100.
- random_state: Controls the randomness of the estimator. Default is None.

8. Random Forest with GridSearchCV

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {

    'n_estimators': [100, 200, 300],

    'max_depth': [None, 10, 20, 30],

    'min_samples_split': [2, 5, 10],

    'min_samples_leaf': [1, 2, 4],

    'bootstrap': [True, False]

}

rf = RandomForestClassifier(random_state=42)

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5,
n_jobs=-1, scoring='accuracy')
```

- n_estimators: Number of trees in the forest.
- max_depth: Maximum depth of the tree.
- min_samples_split: Minimum number of samples required to split an internal node.
- min_samples_leaf: Minimum number of samples required to be at a leaf node.
- bootstrap: Whether bootstrap samples are used when building trees.
- cv: Determines the cross-validation splitting strategy. Default is 5.
- n_jobs: Number of jobs to run in parallel. Default is None.
- scoring: Strategy to evaluate the performance of the cross-validated model. Default is None.

9. Conclusion

The Horse Health Prediction project demonstrates the application of machine learning techniques combined with PCA for dimensionality reduction. Random Forest was found to be the best performing model, achieving an accuracy of 68.19%, with KNN and SVM also performing well. PCA was crucial in reducing the dataset's complexity, enabling faster training and improving generalization. Future work could explore hyperparameter optimization in more detail and incorporate additional models or advanced techniques like deep learning to further improve accuracy.

Code Link:
https://github.com/Gnana-Athityan/ML_horse_health/blob/main/one.ipynb