SAVONIA

THESIS REPORT – DEGREE PROGRAMME IN INTERNET OF THINGS

TECHNOLOGY, COMMUNICATION AND TRANSPORT

Machine Learning and Artificial Intelligence in Data Visualization

AUTHOR/S: Pavel Romanov

Field of Study Technology, communication and transport	
Degree Programme Degree Programme in Information Technology, Internet of Things	
Author(s) Pavel Romanov	
Title of Thesis Machine Learning and Artificial Intelligence in Data Visualization	
Date 28 April 2023	Pages/Appendices 32
Client Organisation /Partners LightningChart Oy	

Abstract

The purpose of this thesis was to create an add-on for the LightningChart JS library to work with machine learning models. The thesis is aimed to be understandable for people unfamiliar with machine learning, while also being suitable for a wide range of applications.

The project was implemented using JavaScript as a programming language and the TensorFlow JS library for machine learning implementation. The LightningChart JS charting library was used to test and visualize the results of the project.

As a result of this thesis, three JavaScript classes representing various supervised machine learning models were created and tested on real use cases. They are suitable for both small and large projects and can solve both regression and classification problems. The results of this thesis are highly customizable and can serve as a base for future work.

Keywords
Machine learning, data visualization, linear regression, logistic Regression, K-Nearest Neighbors, lightningchart js

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Artificial intelligence and machine learning have gained significant importance across various sectors, including healthcare, public safety, banking, and agriculture. Machine learning has revolutionized the analysis of vast amounts of data, simplifying and accelerating the process, thus becoming an indispensable tool in data-driven fields. Among the many domains benefiting from machine learning, data science stands out as a key area.

Data visualization, as a subset of data science, plays a crucial role in transforming complex data structures and patterns into a comprehensible format for human understanding. It finds extensive applications in logistics, science, finance, healthcare, marketing, and education, among other fields. However, the utilization of machine learning methods in data visualization remains a challenge for many individuals, primarily due to the complexity of implementation and the high cost associated with hiring specialized professionals in this domain.

To bridge this gap and simplify the process of working with machine learning for users with no prior experience, this thesis aims to develop an add-on to the LightningChart JS library in collaboration with LightningChart Oy. This add-on will enable users to work with machine learning models directly in their browser, eliminating the need for in-depth knowledge of the field. By integrating machine learning capabilities into a data visualization library, the thesis aims to simplify the usage of machine learning techniques, making them more accessible and user-friendly.

Furthermore, this thesis shows the practical application of machine learning models within the LightningChart JS library by demonstrating real-world use cases. Three machine learning classes: k-nearest neighbors, linear regression, and logistic regression are implemented and tested using LightningChart JS. These use cases highlight the potential of combining data visualization and machine learning, showing how the integration of these technologies can enhance data analysis and decision-making processes. This thesis aims to help users implement machine learning algorithms in their data visualization workflows. The proposed solution allows professionals from different backgrounds to use machine learning in their fields without needing advanced technical skills.

1.1 Structure

The thesis is divided into four chapters, each of which describes different stages of this thesis project.

- Chapters 1, and 2 are the preparation for the implementation of the project. They describe the goals of the project, analysis of the relevance of the project, as well as the necessary background information.

- Chapter 3 demonstrates the classes developed as a result of the thesis project and explains their main methods.

- Chapters 4 and 5 conclude the thesis report by showing the use of implemented classes on real-world examples and evaluating the thesis work.

1.2 Project relevance

Machine learning technology is now more relevant than ever and has a huge number of applications. One of the main goals of this project is to be able to apply it in the largest number of areas of human life. To achieve this, some of the most popular machine learning models are used. The following are some of the possible applications of this project:

- Finance

Forecasting sales: Linear regression can be used to model the relationship between sales and various predictors such as advertising spending, seasonality, and pricing (Ruby 2020).

Stock prediction: Logistic regression can be used to predict whether a stock will go up or down based on historical price movements (Godot 2023). KNN can be used to predict stock prices based on the past price patterns of similar stocks (Chaudhary 2020).

- Marketing

Consumer behavior prediction: Logistic regression can be used to predict whether a customer will buy a product based on various demographic, behavioral, and psychographic factors. KNN can be used to predict customer behavior based on the past behavior patterns of similar customers (Frohböse 2020).

- Security

Fraud identification: Logistic regression can be used to predict the likelihood of fraud based on various factors such as transaction amount, location, and time. KNN can be used to identify fraudulent transactions based on the past transaction patterns of similar customers (Adithyan 2020).

- Other

Spam detection: Logistic regression can be used to classify emails as spam or not based on various features such as sender, subject, and content (Sharma 2018).

Weather forecasting: Linear regression can be used to model the relationship between various meteorological factors such as temperature, humidity, and pressure (Thilakarathne 2020).

2 BACKGROUND INFORMATION

2.1 Machine learning

Machine learning is a field of artificial intelligence that uses algorithms to learn from historical data. While there are many different definitions of machine learning, this is a general description of its main purpose. With the help of machine learning, AI systems can analyze data, memorize information, make predictions, reproduce pre-made models, and choose the most suitable option from the proposed choices. Machine learning is particularly useful in situations that require a large amount of computation, such as bank scoring, analytics in the field of marketing and statistical research, business planning, demographic research, investments, and the identification of fake news and fraudulent sites. In these and other areas, machine learning algorithms can significantly simplify data analysis, speed up decision-making processes, and reduce the potential for human error. Machine learning has become an increasingly important tool for many businesses and organizations, as it can help them to uncover valuable insights, gain a competitive advantage, and make more informed decisions (Redmon 2018).

2.1.1 Types of machine learning

There are many subsets of machine learning, each with its own optimal use case, data shape, and training methods. In this context, four main types of machine learning can be highlighted below.

2.1.1.1 Supervised learning

Supervised machine learning requires a dataset with labeled inputs and desired outputs. Once trained, predictions about new observations can be made. Supervised learning can be divided into two subtypes: regression and classification. In the first case, the output is a continuous value, while in the second case, the output is a discrete value or a probability (Géron 2017, 8-9; Salian 2018).

2.1.1.2 Unsupervised learning

Unsupervised machine learning algorithms use unlabeled data for training. They do not predict the output like supervised learning, but instead explore the dataset, searching for meaningful connections and describing hidden structures in unlabeled data. Depending on the model's objectives, unsupervised learning can be used to group data in different ways (Géron 2017, 10-13; Salian 2018).

The most common application of unsupervised learning is called clustering. Clustering models look for similar data and group it together. Another type of unsupervised learning model is called anomaly detection. These models look for unusual patterns in a dataset. The last example of unsupervised learning is association models. They use several key data point features to predict other features they are associated with (Géron 2017, 10-13; Salian 2018).

2.1.1.3 Semi-supervised learning

Semi-supervised machine learning use both labeled and unlabeled data for training. The training dataset consists of a small amount of labeled data and a huge amount of unlabeled data. The machine learning algorithm groups similar data together, thus helping to label unlabeled data. This method is especially effective when labeling the data and extracting the desired features from the dataset is costly and time consuming (Géron 2017, 13-14; Salian 2018).

2.1.1.4 Reinforcement learning

Reinforcement machine learning works by setting an algorithm with a distinct goal. This algorithm attempts to find the optimal path to a given goal. If it takes actions that are beneficial to the goal, it receives a reward; if it takes action that moves it away from the goal, it receives a punishment. Thus, the algorithm tries to solve the problem by getting as many rewards as possible, while avoiding punishments (Géron 2017, 14-15; Salian 2018).

2.1.2 Choosing machine learning models

Not all types and models of machine learning are suitable for this project. Therefore, when choosing machine learning models, five factors are considered:

- Model popularity
  According to Matthias Döring (2018), supervised learning models are used in more than 50% of cases, with linear regression and logistic regression ranked first and second respectively. Neural networks make third place, followed by decision trees and SVMs. Figure 1 shows the percentage of use of machine learning models in different fields.

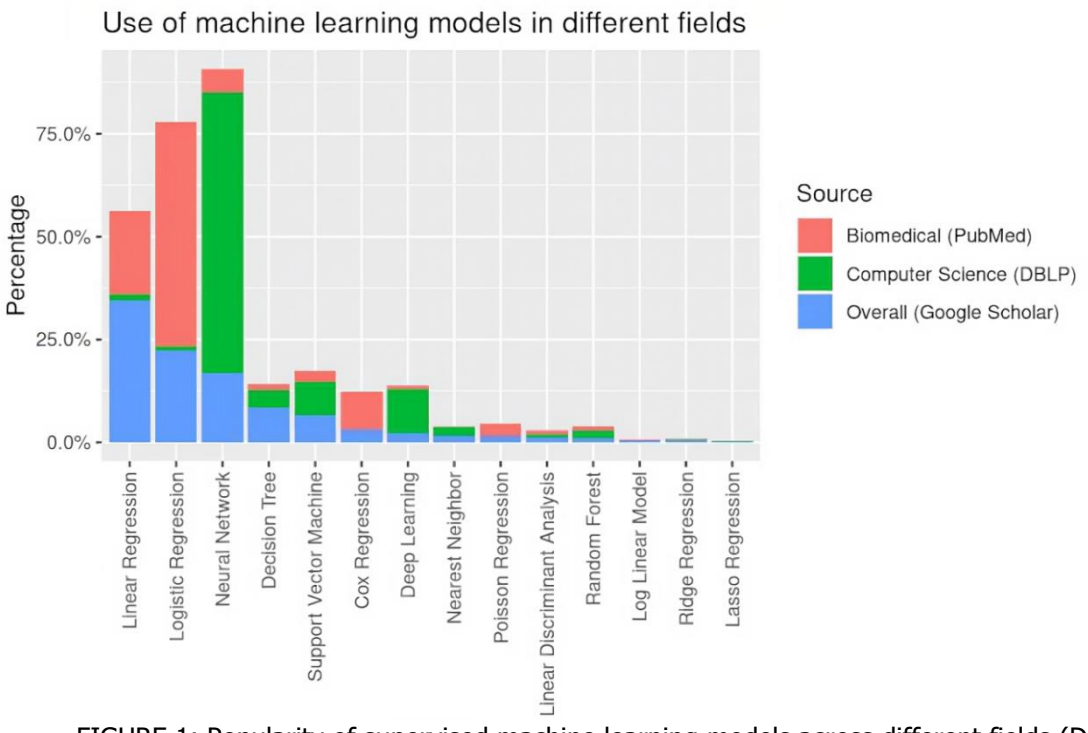

FIGURE 1: Popularity of supervised machine learning models across different fields (Döring 2018).

- Suitability of the model for visualization
  By suitability of the model for visualization, the ability to visualize as many steps in the model as possible is meant. The most important is the ability to visualize the input and output of the model. And secondary is the ability to visualize the parameters and metrics of the model.

- Hardware requirements
  According to Singh (2019), a lot of computing power can be required to train a machine learning model. High performance GPUs and CPUs may be required to complete some tasks in a reasonable amount of time. Since the LightningChart JS library can run on a wide range of devices from low-end to high- end, machine learning models with high hardware load are excluded from selection.

- Amount of input
  Since the goal of this project is to make machine learning models easier to use, the user should not be overwhelmed by the many functions and parameters. Therefore, it is desirable to minimize the amount of input required from the user.

- Complexity of the model
  What stops many from using machine learning, is the difficulty of understanding its algorithms. Thus, being able to understand the learning process and how the model generates its outcome plays an important role in building trust in machine learning. That is why an important factor in choosing a model for this project is the ability of the model user to understand it and explain the principles of the algorithm to others.

2.1.3 Final model selection

Based on all the factors considered, author decided to implement the project using the two most popular supervised learning models: linear regression and logistic regression. These models meet all the necessary criteria and are chosen specifically to demonstrate both regression and classification types of supervised learning.

In addition, it was also decided to include the K-Nearest Neighbors algorithm in the project. This is a simple and versatile supervised learning algorithm that can handle both regression and classification problems. It works well with small amounts of data and is chosen to introduce the user to machine learning.

2.2 Algorithms

This section explains the machine learning algorithms used in this project.

2.2.1 K-Nearest Neighbors algorithm

The basic principle on which the K-Nearest Neighbors algorithm works is the assumption that similar things exist next to each other. This allows KNN algorithm to find patterns between features and labels (independent and dependent data). KNN finds distances between features of query data and features of data points from the prepared dataset. Then it sorts calculated distances from smallest to largest and selects top K data points closest to query point. Further action of the algorithm is different depending on the type of problem. In case of regression, the algorithm calculates the mean of the labels of the nearest K data points. And in case of classification, the algorithm returns the most frequent label of the nearest K data points (Harrison 2018). Figure 2 shows the algorithm used in the KNN model.

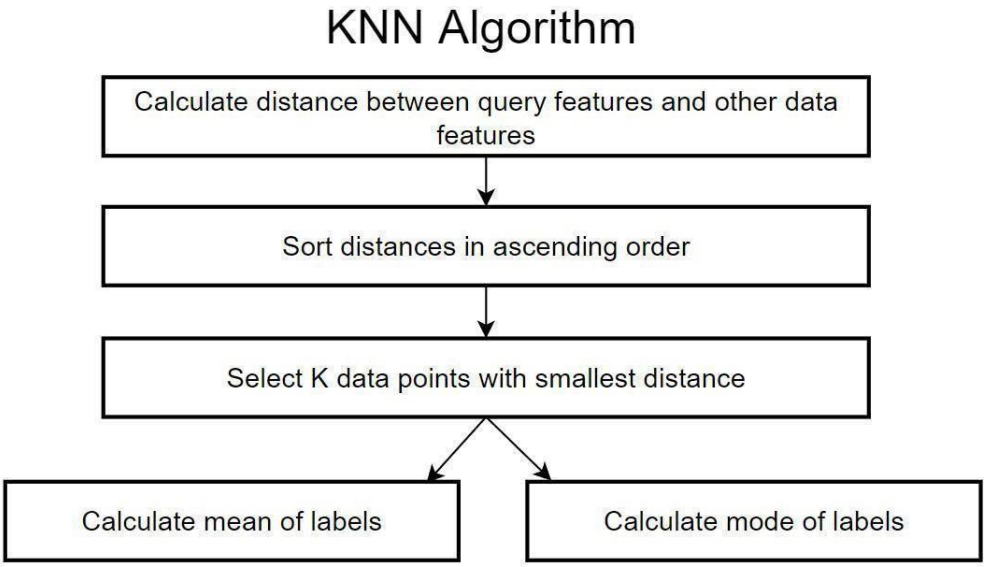

FIGURE 2: K-Nearest Neighbors algorithm.

According to Harrison (2018), to select the appropriate K value, KNN algorithm should be executed several times. This helps to choose the value of K which reduces the number of errors and increase the accuracy of the algorithm. K-Nearest Neighbors is a simple and versatile algorithm that does not require building a model and tuning many parameters. But the main disadvantage of KNN is low performance with a large amount of data.

2.2.2 Solving linear regression with gradient descent algorithm

To implement linear regression and logistic regression, the gradient descent algorithm is be used. This is a popular algorithm that is used in many machine learning models.

According to Montgomery (2021, 4), linear regression is a regression model that attempts to find a relationship between a pair of variables by fitting a linear equation to historical data. For example, the Figure 3 shows a linear function that fits perfectly into the observed dataset. This is what the desired result of the gradient descent algorithm looks like.

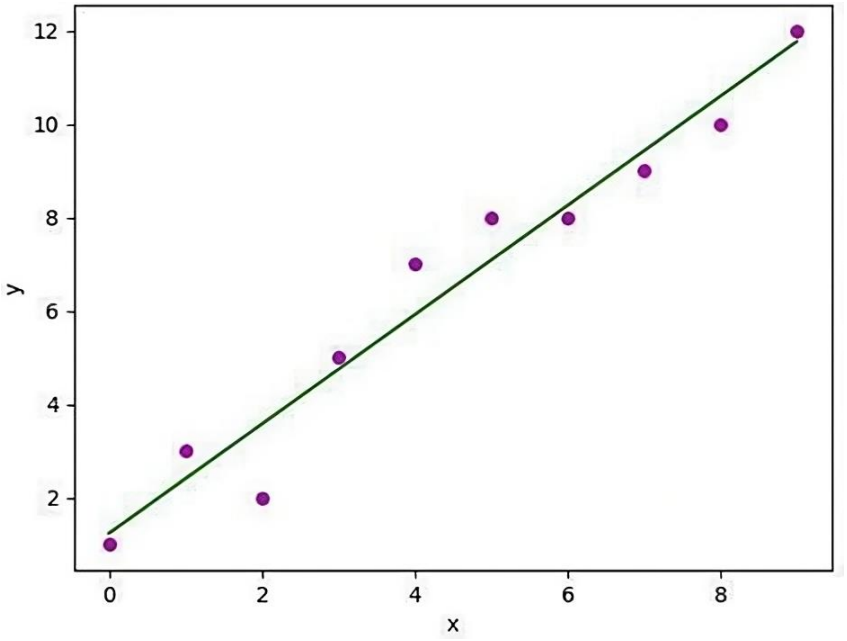

FIGURE 3: Linear function that fits historical data (Karunakaran 2020)

Gradient descent algorithm starts with a random linear function and improves it until the best fitting function is found.

$MSE = \frac{1}{n} * \sum_{i-1}^{n} (actual - forecast)^2$ (Binieli 2018, 1)

equation is used to adjust function. MSE is the average squared difference between the guess value and the actual value (Binieli 2018; Menon 2018).

The MSE needed to determine how well the current linear function fits the data. The lower the MSE, the closer the function is to the ideal result. Let's assume that the linear function looks like f(x) = mx + b, in which case such values of m and b should be chosen so that the MSE of the resulting function is as close as possible to the minimal MSE value. So, to understand how far the current function is from the ideal one, it is necessary to find the minimum value of MSE. This can be achieved by differentiating the MSE and setting it equal to 0. And since in the case of linear regression MSE depends on two variables, two partial derivatives of MSE function should be computed value (Binieli 2018; Menon 2018).

The first partial derivative of MSE

$\frac{\partial MSE}{\partial m} = \frac{1}{n} * \sum_{i=1}^{n} \big(2 * (y_i - (mx + b)\big) * (-x_i))$ (Polyanin 2008, 2)

should be calculated with respect to m, and the second partial derivative of MSE

$\frac{\partial MSE}{\partial b} = \frac{1}{n} * \sum_{i=1}^{n} \big(2 * (y_i - (mx + b)\big) * (-1))$ (Polyanin 2008, 3) with respect to b.

Computing partial derivatives reveals two important facts about the current guess. The first fact is the distance of current guess MSE to 0. The smaller this distance, the closer the current guess is to the optimal values of m and b. And the second fact is the sign of partial derivative. Knowing the sign of partial derivatives, it becomes clear in which direction to change the values of m and b (Géron 2017, 120; Menon 2018).

Now that the current guess has been analyzed using partial derivatives, the m and b values can be adjusted. To do this, the Gradient Descent algorithm uses the so-called learning rate. The learning rate is a parameter that determines the step size at which the model parameters are updated during training. With it, the values of m and b are adjusted using the following formulas:

$m \coloneqq m - learningRate * \frac{\partial MSE}{\partial m}$ (Menon 2018, 4) for m and

$b \coloneqq b - learningRate * \frac{\partial MSE}{\partial m}$ (Menon 2018, 5) for b.

Further, the algorithm repeats the above actions, adjusting the MSE value. It should be noted that it is rarely possible to achieve the minimum value of MSE. Therefore, the gradient descent algorithm should be stopped when it no longer makes meaningful changes to the MSE value (Menon 2018). Figure 4 shows the gradient descent algorithm used to solve the linear regression model.

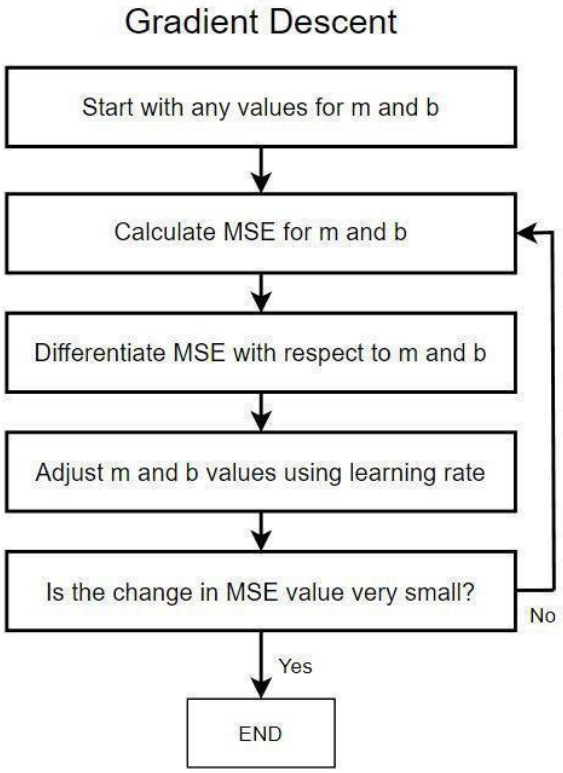

FIGURE 4: Gradient descent algorithm.

2.2.3 Solving logistic regression with gradient descent algorithm

Logistic regression is a supervised machine learning classification model that evaluates the probability of an instance belonging to a particular class by finding the relationship between independent and discrete variables. Although a gradient descent algorithm is also used to implement this model, there are several differences in its implementation compared to linear regression (Géron 2017, 144).

The first difference is that independent data must be represented in discrete form. For example, when calculating the probability of rain, rainy weather can be represented as 1, and non-rainy weather as 0 (Géron 2017, 144-145).

The second difference is that in the case of logistic regression, a sigmoid function is used instead of a linear function. This is because a linear function does not fit well on discrete data for two reasons. The first reason is that since the linear function goes beyond 0 and 1 on the y-axis, it is impossible to get the probability as an output. The second reason is that leverage points (data points with an unusual independent value) can affect the linear function too much (Géron 2017, 144-145).

It is also necessary to consider that in the case of logistic regression, the relationship between MSE and the values of m and b is not a parabola. Therefore, updating the learning rate depending on the change in MSE may result in locating a local minimum instead of a global one. This leads to the fact that the algorithm not always finds the optimal value of m and b. To avoid this problem Cross Entropy (cost function) is used instead of MSE.

$Cross\ Entropy = -\left(\frac{1}{n}\right) \sum_{i=0}^{n}(actual * \log(guess) + (1 - actual) * \log(1 - guess))$ (Brownlee 2019, 6) is a loss function that is used to quantify the difference between two probability distributions (Géron 2017, 145-146). Figure 5 shows the difference between linear regression and logistic regression models.

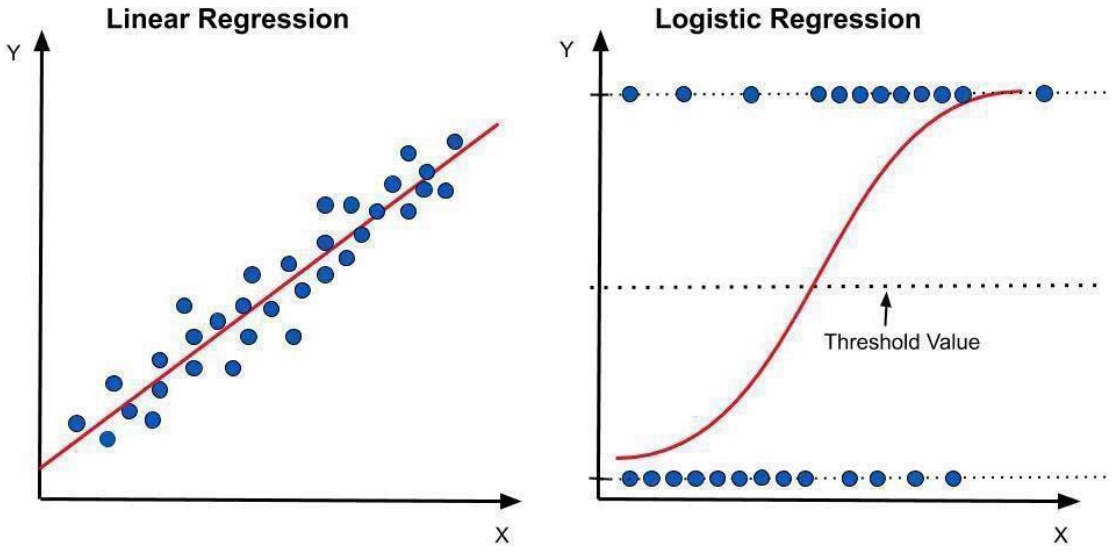

FIGURE 5: Linear regression and logistic regression examples (Mehta 2020)

3 IMPLEMENTATION

3.1 TensorFlow JS

To implement machine learning models in JavaScript, the TensorFlow JS library is chosen. It allows to develop machine learning models in JavaScript and use machine learning directly in the browser or in Node.js. The reason for choosing this library is because it allows the user to train a machine learning model using their own resources. This means that a machine learning model on a website can be trained using a client's computer with data generated in real time from that user's interaction with the site. Also, instead of constantly exchanging data between the client and the server, TensorFlow JS allows to use the algorithm in real time during its training (Bicknese 2020).

The main concept to know about TensorFlow is that it uses its own date type called tensor. According to the TensorFlow JS API, tensor is a set of values shaped into an array of one or more dimensions with the following main properties:

- dtype - The data type
- rank - The number of dimensions
- shape - The size of each dimension

A Tensor can be created from following data types:

- float32
- bool
- int32
- complex64
- string

3.2 KNN implementation

3.2.1 KNN code

This code contains an implementation of the K-Nearest Neighbors algorithm.

```
const tf = require('@tensorflow/tfjs');

// Class constructor
class KNN {
  constructor(features, labels, k,) { this.features = features this.labels = labels
    this.k = k
  }
    // KNN algorithm using standartizations scaling
    standartizationKNN(testPoint)  {
// Converting variables to tensor
const tensorFeatures = tf.tensor(this.features) const tensorLabels =
tf.tensor(this.labels) const predictionPoint = tf.tensor(testPoint)

// Calculating mean and vatiance for standartization
const { mean, variance } = tf.moments(this.features, 0);
// Standartization of query point
const scaledPrediction = predictionPoint.sub(mean).div(variance.pow(0.5));
```

```
    return (

      tensorFeatures
        // STANDARTI ATION
        .sub(mean)
        .div(variance.pow(0.5))
        // DISTANCE CALCULATION
       .sub(scaledPrediction)
       .pow(2)
       .sum(1)
       .pow(0.5)
       // CONCAT FEATURES AND LABELS TOGETHER
       .expandDims(1)
       .concat(tensorLabels, 1)
       // UNSTACK TO A JA ASCRIPT ARRAY
      .unstack()
      // SORT
     .sort((tensorA, tensorB) => (tensorA.arraySync()[0] > tensorB.arraySync()[0]?1 : -1))


      // TAKE TOP K RECORDS
      .slice(0, this.k)
      // A ERAGE OF K OF LABELS CLOSEST TO THE PREDICTION POINT
)     .reduce((acc, pair) => acc + pair.arraySync()[1], 0) / this.k


  }

  // KNN algorithm using normalization scaling
  normalizationKNN(testPoint)  {

  // Converting variables to tensor
  const normalizedArray = this.minMax(this.features, this.features[0].length, testPoint)
  const tensorFeatures = tf.tensor(normalizedArray[0]) const scaledPoint =
  tf.tensor(normalizedArray[1]) const tensorLabels = tf.tensor(this.labels)

  return ( tensorFeatures
      // DISTANCE CALCULATION
      .sub(scaledPoint)
      .pow(2)
      .sum(1)
      .pow(0.5)
      // CONCAT FEATURES AND LABELS TOGETHER
      .expandDims(1)
      .concat(tensorLabels,  1)
      // UNSTACK TO A JA ASCRIPT ARRAY
      .unstack()
      // SORT
     .sort((tensorA, tensorB) => (tensorA.arraySync()[0] > tensorB.arraySync()[0]?1 : -1))

      // TAKE TOP K RECORDS
      .slice(0, this.k)
      // A ERAGE OF K OF LABELS CLOSEST TO THE PREDICTION POINT
      .reduce((acc, pair) => acc + pair.arraySync()[1], 0) / this.k

)

 }
```

```
    // Min Max Normalization
    minMax(data, featuresCount, point) {

      const dataArr = []
      for (let i = 0; i < data.length; i++){
        dataArr.push([])
      }

      const pointArr = []

      for (let i = 0; i < featuresCount; i++) {
        const column = (arr, n) => arr.map((row) => row[n])

        const min = Math.min.apply(Math, column(data, i))
        const max = Math.max.apply(Math, column(data, i))

        pointArr.push((point[i] - min) / (max - min))

        for (let j = 0; j < data.length; j++) {
          dataArr[j].push((data[j][i] - min) / (max - min))
        }
      }
      return [dataArr, pointArr]
    }
  }

module.exports = KNN;
```

3.2.2 KNN code explanation

This is a JavaScript class containing a K-Nearest Neighbors algorithm. It contains two KNN methods that differ in the type of data scaling.

KNN class has following properties:

- features – independent data.
- labels - dependent data.
- k - number of data points used for calculation.

The "standardizationKNN" method implements the KNN algorithm using standardization scaling

$x_{stand} = \frac{x - mean(x)}{standart\ deviation(x)}$ (Liu 2022, 7).

Standardization is used when data follows a Gaussian distribution. Also, standardization does not affect outliers, since it has no bounding range. In the case of KNN, this can be useful so that outliers do not affect the calculations too much (Bhandari 2020; Liu 2022).

The "normalizationKNN" method implements the KNN algorithm using normalization scaling

$x_{norm} = \frac{x - min(x)}{max(x) - \min(x)}$ (Liu 2022, 8).

Normalization is used when the data does not have Gaussian Distribution, or the data distribution is unknown. Normalization is applied in "minMax" method (Bhandari 2020; Liu 2022).

3.3 Linear regression implementation

3.3.1 Linear regression code

This code contains an implementation of the gradient descent algorithm for a linear regression model.

```
const tf = require('@tensorflow/tfjs');
const _ = require('lodash');

class LinearRegression {
  constructor(features, labels,
  options) {
    // independent data
    this.features = this.processFeatures(features);
    // dependent data
    this.labels = tf.tensor(labels);
    // array with history of mean squared error
    this.mseHistory = [];
    // array with history of learning
    rate this.learningRateHistory =
    []; this.options = Object.assign(
      { learningRate: 0.1, iterations:
      100 }, options
    );
    // m and b values
    this.weights = tf.zeros([this.features.shape[1], 1]);
  }
 gradientDescent(features, labels) {
// Calculating the slope of MSE with respect to m and b
(features* (features*weights)-labels)/observations
const currentGuesses = features.matMul(this.weights);
const differences = currentGuesses.sub(labels);

const slopes = features
  .transpose()
  .matMul(differences)
  .div(features.shape[0]);

// Updating m and b
this.weights = this.weights.sub(slopes.mul(this.options.learningRate));
}

// Training loop
train() {
  // Calculating number of batches
  const batchQuantity = Math.floor(
    this.features.shape[0] / this.options.batchSize
  );
```

```
  for (let i = 0; i < this.options.iterations; i++) {
    // Iterating through batches of data
    for (let j = 0; j < batchQuantity; j++) {
      const startIndex = j * this.options.batchSize; const {
      batchSize } = this.options;
      // Slicing features
      const featureSlice = this.features.slice( [startIndex, 0],
        [batchSize, -1]
      );
      // Slicing labels
      const labelSlice = this.labels.slice([startIndex, 0],
      [batchSize, -1]);

      this.gradientDescent(featureSlice,  labelSlice);
    }

     this.recordMSE(); this.updateLearningRate();
   }

   }
   // Prediction
   predict(observations)  {
     const prediction =
 this.processFeatures(observations).matMul(this.weights).dataSync();  return
     Array.from(prediction)[0]
   }

   //Testing by calculating the coefficient of determination
   test(testFeatures, testLabels)  {
     testFeatures = this.processFeatures(testFeatures); testLabels =
     tf.tensor(testLabels);

     const predictions = testFeatures.matMul(this.weights);

     const res = testLabels

    .sub(predictions)
    .pow(2)
    .sum()
    .get();
  const tot = testLabels
    .sub(testLabels.mean())
    .pow(2)
    .sum()
    .get();

  return 1 - res / tot;
}

   // Standartization applied to features
processFeatures(features) { features = tf.tensor(features);
  features = tf.ones([features.shape[0], 1]).concat(features, 1);
```

```
  if (this.mean && this.variance) {
    features = features.sub(this.mean).div(this.variance.pow(0.5));
  } else {
    features = this.standardize(features);
  }

  return features;
}

standardize(features) {
  const { mean, variance } = tf.moments(features, 0);

  this.mean = mean;
  this.variance = variance;

  return features.sub(mean).div(variance.pow(0.5));
}

// Recording MSE for updating learning rate
recordMSE() {

        const mse =
       this.features
     .matMul(this.weig
           hts)
       .sub(this.labels)
       .pow(2)
       .sum()
       .div(this.features.shape[0])
       .dataSync()[0,0];

       this.mseHistory.push(mse);

  }

  // Updating learning rate depednding on mse change
  updateLearningRate() {
    if (this.mseHistory.length < 2) {
      return;
    }


    if (this.mseHistory[this.mseHistory.length-1] >
this.mseHistory[this.mseHistory.length-2]) {
       this.options.learningRate /= 2;
    } else {
       this.options.learningRate *= 1.05;
    }
    this.learningRateHistory.push(this.options.learningRate);
}
}

module.exports = LinearRegression;
```

3.3.2 Linear regression code explanation

This is a JavaScript class for solving the Linear Regression with Gradient Descent algorithm. The following is a brief description of its methods.

The "processFeatures" and "standardize" methods are used to apply standardization on features. The "train" method is used to create a training loop and split the data into batches. Considering that in the following example (4.4) the model is trained on a small dataset, dividing the dataset into batches provides stable gradient descent convergence without increasing the training time and not being heavily dependent on computer resources. The "gradientDescent" method is used to calculate the slope and update m and b values as shown in 2.2.2. The "updateLearningRate" and "recordMSE" methods are used to update learning rate. Knowing the difference between the current and previous MSE value, the efficiency of the algorithm can be increased by adjusting the learning rate. The "test" method is used for testing the model by calculating the coefficient of determination. The "predict" method is used to get predictions from trained model.

3.4 Logistic regression implementation

3.4.1 Logistic regression code

This code contains an implementation of the gradient descent algorithm for a logistic regression model.

```
const tf = require('@tensorflow/tfjs'); const _ = require('lodash');

class LogisticRegression {
  constructor(features, labels, options) {
    // independent data
    this.features = this.processFeatures(features);
    // dependent data
    this.labels = tf.tensor(labels);
    // array with history of cost
    this.costHistory = [];

    this.options = Object.assign(
      { learningRate: 0.1, iterations: 1000, decisionBoundary:
      0.5 }, options
    );

    // m and b values
    this.weights = tf.zeros([this.features.shape[1], 1]);
  }

  gradientDescent(features, labels) {
    // Calculating the slope of MSE with respect to m and b
    const currentGuesses = features.matMul(this.weights).sigmoid();
    const differences = currentGuesses.sub(labels);
```

```
  const slopes = features
    .transpose()
    .matMul(differences)

    .div(features.shape[0]);
  // Updating m and b
  this.weights = this.weights.sub(slopes.mul(this.options.learningRate));
}
     // Training loop
train() {
  // Calculating number of batches
  const batchQuantity = Math.floor(
    this.features.shape[0] / this.options.batchSize
  );

  for (let i = 0; i < this.options.iterations; i++) {
    // Iterating through batches of data
    for (let j = 0; j < batchQuantity; j++) {
      const startIndex = j * this.options.batchSize; const { batchSize }
      = this.options;
      // Slicing features
      const featureSlice = this.features.slice( [startIndex, 0],
        [batchSize, -1]
      );
      // Slicing labels
      const labelSlice = this.labels.slice([startIndex, 0], [batchSize, -
      1]);

      this.gradientDescent(featureSlice,  labelSlice);
    }
    this.recordCost(); this.updateLearningRate();
            }
          }
          // Prediction(probability)
          predict(observations)  {
            return this.processFeatures(observations)
              .matMul(this.weights)
              .sigmoid()
              .dataSync()[0];
          }
          // Prediction(classification)
          predictRound(observations)  {
            return this.processFeatures(observations)
              .matMul(this.weights)
              .sigmoid()
              .greater(this.options.decisionBoundary)
              .cast('float32')
              .dataSync()[0];
          }
            //Testing returns percentage of correct predictions
          test(testFeatures, testLabels)  {
            const predictions = this.predict(testFeatures); testLabels =
            tf.tensor(testLabels);
```

```
            // Number of incorrect predictions
            const incorrect = predictions
              .sub(testLabels)
              .abs()
              .sum()
              .dataSync();
              return (predictions.shape[0] - incorrect) /
            predictions.shape[0];
          }
        // Standartization applied to features
        processFeatures(features) { features = tf.tensor(features);

          if (this.mean && this.variance) {
            features = features.sub(this.mean).div(this.variance.pow(0.5));
          } else {
            features = this.standardize(features);
          }

          features = tf.ones([features.shape[0], 1]).concat(features, 1);

          return features;
        }

        standardize(features) {
          const { mean, variance } = tf.moments(features, 0);

          this.mean = mean;
          this.variance = variance;

          return features.sub(mean).div(variance.pow(0.5));
        }

        // Recording cross entropy history to updatae learning rate
        recordCost() {
          const guesses = this.features.matMul(this.weights).sigmoid();
    const termOne = this.labels.transpose().matMul(guesses.log());

    const termTwo = this.labels
      .mul(-1)
      .add(1)
      .transpose()
      .matMul( guesses
          .mul(-1)
          .add(1)
          .log()
      );

    const cost = termOne
      .add(termTwo)
      .div(this.features.shape[0])
      .mul(-1)
      .dataSync()[0,0];

    this.costHistory.unshift(cost);
  }
```

```
    // Updating learning rate
    updateLearningRate()  {
      if (this.costHistory.length < 2) {
        return;
      }

      if (this.costHistory[0] > this.costHistory[1]) {
        this.options.learningRate /= 2;
      } else {
        this.options.learningRate *= 1.05;
      }
    }
  }


  module.exports = LogisticRegression;
```

3.4.2 Logistic regression code explanation

This JavaScript class is designed to solve the Logistic Regression using the gradient descent algorithm. Although it shares similarities with the Linear Regression class, there are some significant differences. The first change is in the "gradientDescent" method, which utilizes a sigmoid function instead of a linear one. Additionally, a different loss function called Cross Entropy is used instead of MSE, as explained in section 2.2.3. Due to this difference, the "recordMSE" method is replaced by the "recordCost" method.

Since Logistic Regression is a classification model, the approach to testing the model differs from regression models. Therefore, the "test" method calculates the percentage of correct predictions. It is worth noting that the addition of a new option to the parameters, called decisionBoundary, is the next major change in this class. This parameter is set to 0.5 by default. Therefore, predictions above this threshold are classified as 1, and those below as 0. However, in some instances, this parameter may need to be adjusted based on the model's purpose. For example, in situations where human life is at risk, even the slightest chance of danger should be classified as dangerous. In such cases, the decisionBoundary parameter can be set to a lower value.

Finally, this class has two prediction methods. The "predict" method provides the probability of an event occurring, while the "predictRound" method rounds the probability and classifies the event based on the decisionBoundary parameter. These features are unique to Logistic Regression and make it an effective tool for solving classification problems.

4 TESTING

This chapter demonstrates the work of the created machine learning classes using real-world examples. To do this, using the LightningChart JS library, four applications are created that demonstrate both machine learning algorithms and visualization of the results of their work. The code of these applications is available on the GitHub of the author of the thesis in the thesis-appendix repository.

4.1 LightningChart JS

LightningChart JS is a WebGL-based JavaScript charting library. It is capable of displaying massive amounts of data inside browser or web application while maintaining excellent performance with GPU acceleration & WebGL rendering. These qualities make this library well suited for use with machine learning models, which often require huge amounts of data and are highly dependent on both GPU and CPU performance.

4.2 KNN classification app example

The first application shows the use of the KNN class in a classification problem. It classifies data points based on their coordinates. Figure 6 shows the interface of application that uses a dataset with points on two-dimensional coordinates classified by two colors: red and blue.

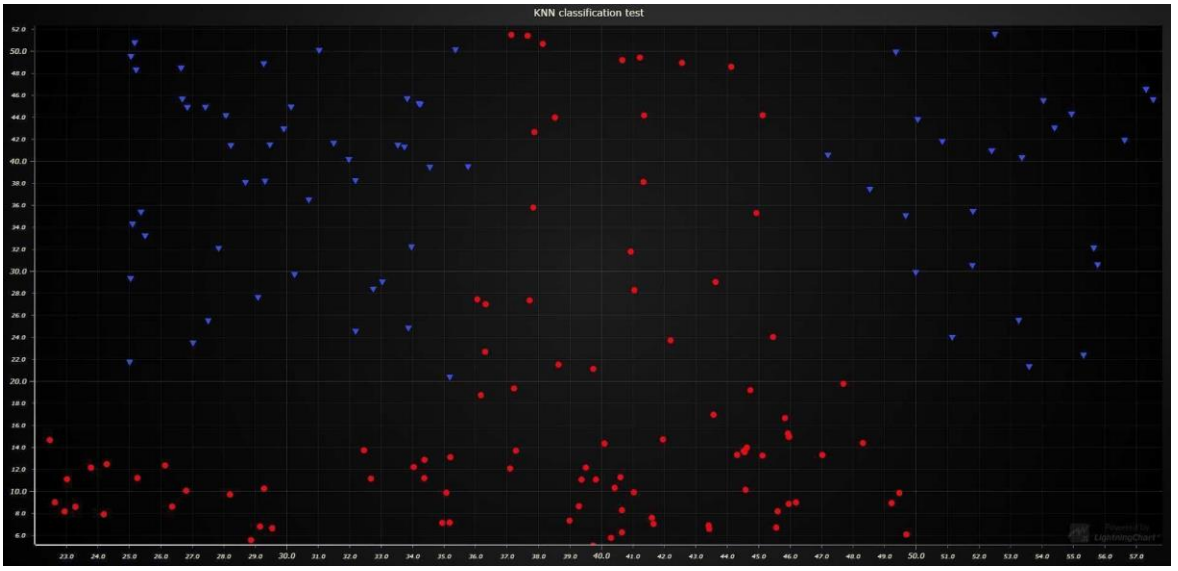

FIGURE 6: KNN classification application.

When the coordinates are clicked, a new point is created and classified by color depending on its position in the coordinates. Figure 7 shows the newly added points, highlighted in green for convenience. Testing has shown that points are successfully classified depending on their location on the coordinate system.

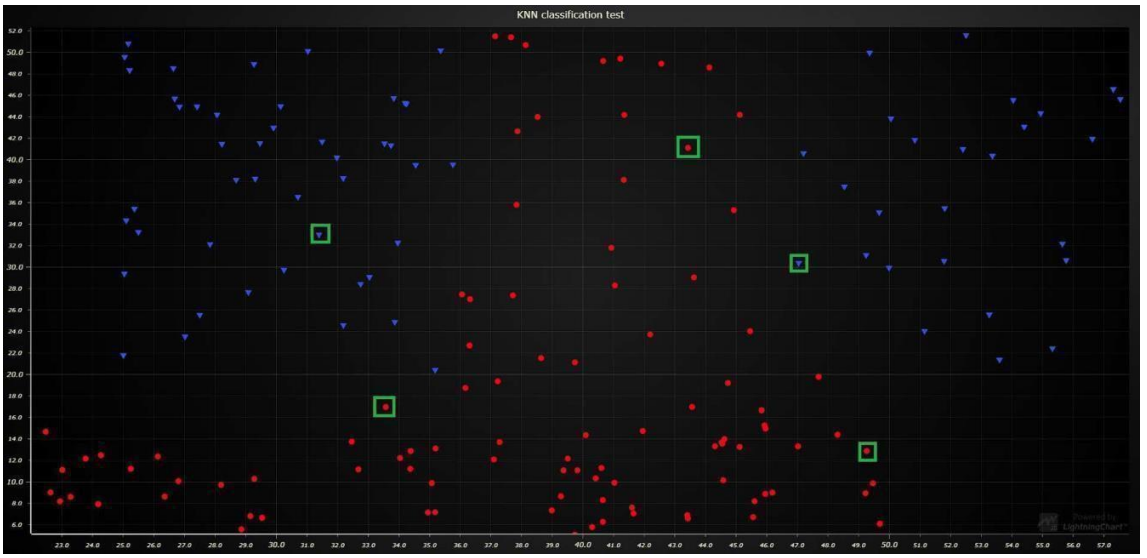

FIGURE 7: KNN classification app (new classified data points).

In this application, coordinates are used as features and red and blue colors are represented as 0 and 1 and used as labels. Using the KNN class, the user only needs to take a few simple steps to implement the algorithm in their application.

1. Fill two arrays of arrays: one with features and one with labels.

```
const features = []
const labels = []

// Assign data to features and labels
for (let i = 0;i<data.length;i++){
    if (data[i].color ==
        "red"){
        features.push([data[i].x,data[
        i].y]) labels.push([0])

    }
    else{
        features.push([data[i].x,data[
        i].y]) labels.push([1])

    }
}
```

2. Create an object from the KNN class using features, labels, and k as parameters.

```
const testKNN = new KNN(features, labels, 10);
```

3. Call a "normalizationKNN" method using query features as parameters.

```
const result = testKNN.normalizationKNN([mouseX,mouseY])
```

4. Since the result of the algorithm is the average number of k labels, to classify a query point the result must be rounded.

```
// Assign result to point series
if (Math.round(result) == 0){
    redPoints.add({x:mouseX,y:mous eY})
}
else{
    bluePoints.add({x:mouseX,y:mouseY})
}
```

4.3 KNN regression app example

The second application shows the use of the KNN class in a regression problem. It predicts the price of a house based on its size and coordinates. Figure 8 illustrates the application interface, which includes two maps and a UI panel. The smaller map located in the lower right-hand corner displays a map of Seattle. When the user clicks on this map, the selected area of the city is zoomed in on the large map on the left-hand side. The blue points on the left-hand map indicate house objects that contain the house's characteristics and its price. The location of these points on the chart corresponds to their actual location on the map.

To select the size of the house, the user can use the slider located in the upper right-hand corner. Clicking on the map generates a data point with the estimated price of a house of the selected size at that location.

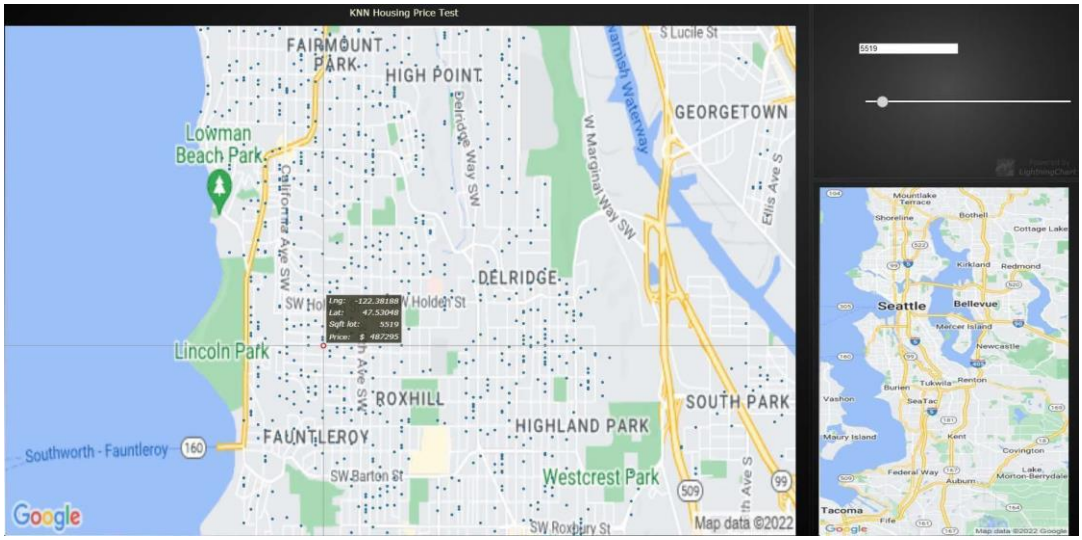

FIGURE 8: KNN regression application.

This app is built using the Google Maps API. The features are latitude, longitude, and the size of the house in square feet. House prices in Canadian dollars are used as labels. The algorithm uses only data points that are visible on the left map to speed up calculations. Also, standardization scaling is used in this application to limit the influence of very expensive houses on calculations. The use of the KNN class is the same as the example described in 4.2. Testing has shown that the predicted price is very close to the real data in various areas of the city. It is expected that the denser the data points are located to the input data point, the more accurate the prediction is.

4.4 Linear regression app example

The third application shows the use of the linear regression class. It predicts a car's MPG based on its weight and horsepower. Such an application can be useful for car manufacturers to predict certain parameters of a car. Figure 9 illustrates the application interface, which consists of three charts: the left chart is used to create a query and display information, while the two charts on the right allow the user to track the performance of the model and adjust its parameters by displaying changes in MSE and learning rate over time.

The two-dimensional chart on the left consists of two axes: the x-axis represents horsepower, and the y-axis represents weight. When a point is clicked on the chart, it creates a data point with the corresponding parameters and the predicted MPG. Testing has shown that even using a small dataset, the predicted characteristics of cars are close to real ones.

For instance, if the user notices that the MSE does not change for a certain number of iterations, they can reduce the number of iterations to reduce the training time of the model. The two charts on the right allow the user to monitor the performance of the model and make necessary adjustments to improve its accuracy.

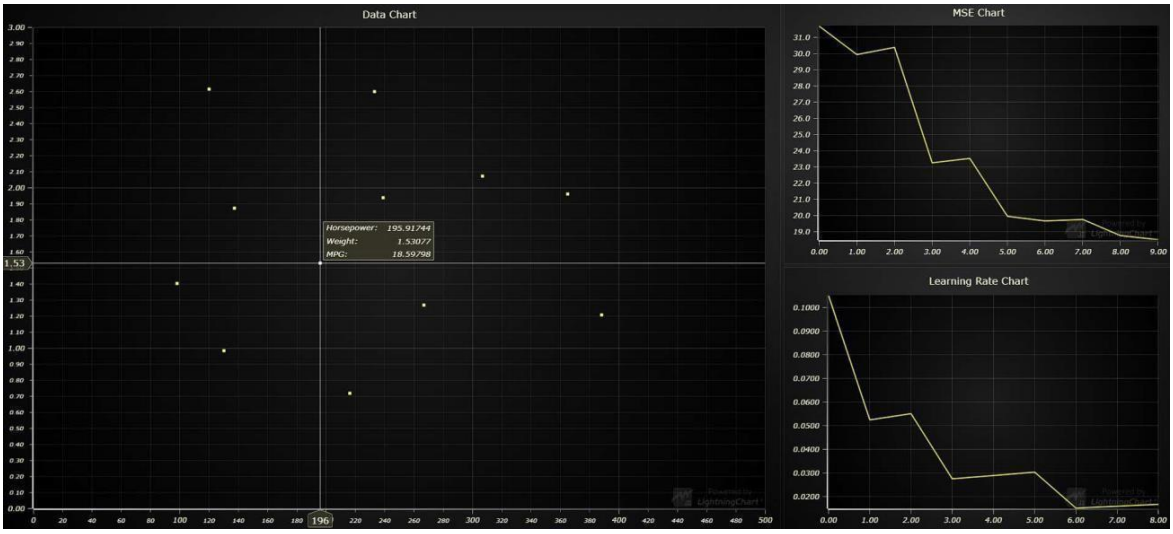

FIGURE 9: Linear regression application.

An example of using the Linear Regression class on this application:

1. Fill two arrays of arrays: one with features and one with labels.

```
const features = []
const labels = []
// Assigning data to features and labels
for (let i = 0; i<data.length;i++){
  features.push([data[i].horsepower,data[i].weight])
  labels.push([data[i].mpg])
}
```

2. Create an object from the Linear Regression class.

```
// Creating linear regression object
const regression = new
      LinearRegression(features, labels, {
      learningRate: 0.1,
      iterations: 10,
      batchSize: 10
         });
```

3. Call "train" method to train the model.

```
// Training the model
regression.train();
```

4. Get mseHistory and learningRateHistory and add them to the right charts.

```
// Getting model properties
const mse = regression.mseHistory
const learningRate = regression.learningRateHistory
// Adding data points to MSE and LR charts

for (let i = 0; i <
        mse.length;
        i++){
        MSEseries.add
        ({x:i,y:mse[i
        ]})
    }

    for (let i = 0; i < learningRate.length; i++){
        LRseries.add({x:i,y:learningRate[i]})
    }
```

5. Call "predict" method to get the result of the algorithm.

```
const prediction = regression.predict([[mouseX, mouseY]])
```

4.5 Logistic regression app example

The last application shows the use of the logistic regression class. It simulates real-time data, based on which it makes an assumption about the probability of an event occurring. For this program, a dataset is created with observations of some fictitious machine at the factory. This dataset proves that the machine breaks down under unknown circumstances. The purpose of the application is to find the relationship between the parameters of the machine and its breakdown. Then, monitor the operation of the machine in real-time and predict the probability of breakdown occurring.

Figure 10 shows the interface of the application. On the bottom chart, the operation of the machine can be observed. The x-axis is the iterations of the work, and the y-axis is the speed of the machine. By hovering the mouse over a data point, a table of parameters in the current iteration can be seen. In this example, the machine has only two parameters: speed and temperature.

The top chart shows the probability of the machine breaking down for each of its iterations. This probability is predicted with the arrival of new data. Testing showed that when data arrives in real-time, the application instantly calculates the probability of a breakdown and visualizes it. This allows factories to take immediate action and avoid emergencies.

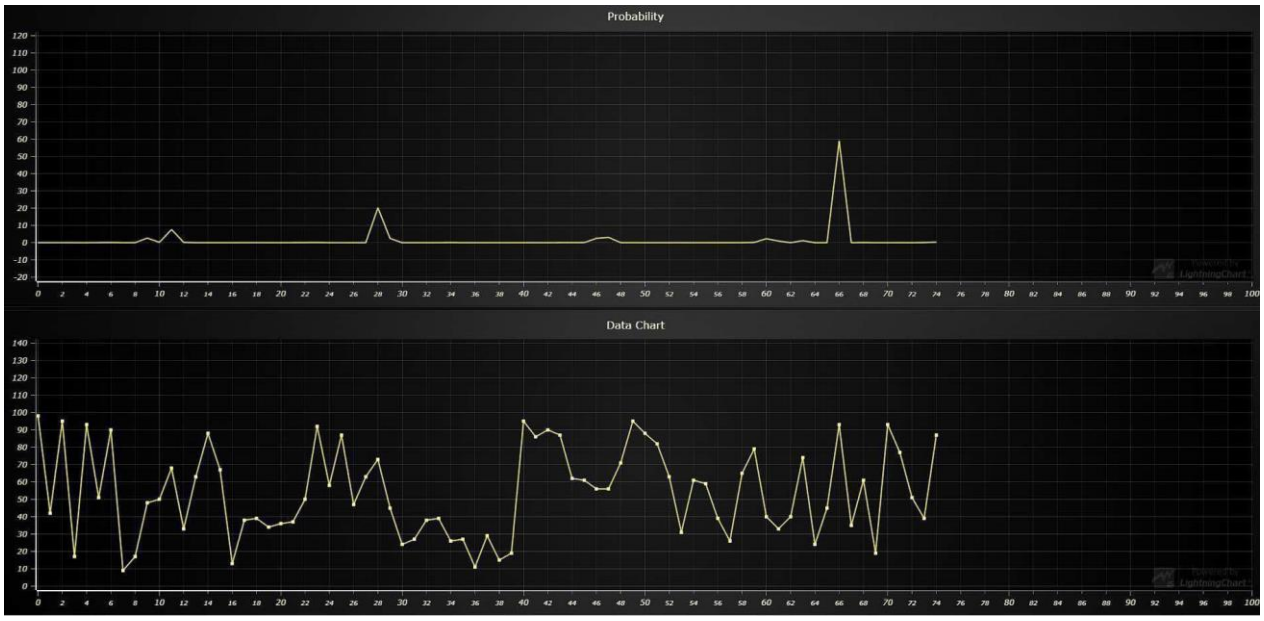

FIGURE 10: Logistic regression application.

The use of the Logistic Regression class is no different from the use of the Linear Regression described in 4.4. Except for three small differences:

1. When the object is created, the decisionBoundary can be included in the options parameter. This sets the threshold for the classification.

2. Depending on the desired output, two methods can be used – "predict" for probability and "predictRound" for classification.

3. To monitor model performance, "recordCost" must be used instead of "recordMSE" for reasons described in 2.2.3.

5 DISCUSSION

The thesis demonstrated the principles of machine learning models that are most suitable for use with data visualization. The algorithms used for their implementation were also analyzed. Then, based on their logic, machine learning classes were developed in the JavaScript programming language using the TensorFlow JS library. These classes significantly improve the quality of work with machine learning methods, on the one hand, simplifying difficult algorithms, and on the other hand, maintaining the flexibility in customizing models.

Furthermore, by employing the LightningChart JS library, several applications were created for testing classes on real-world examples. All three classes (KKN, linear-regression, logistic regression) exhibited impressive performance in the browser and on low-spec hardware. Additionally, these classes have shown excellent compatibility with the LightningChart JS library. LightningChart JS is not only capable of visualizing the training dataset, features, and labels of the model, but also various error metrics. As a result, users can better understand the principles of operation of models and algorithms and tune the model for their needs.

While this enhancement to the LightningChart JS library is a fantastic entry point into machine learning, it provides a toolkit for more advanced users as well. Users not familiar with machine learning can use the KNN class, which requires little to no knowledge in this topic. If the user understands the concept of independent and dependent data, they can, with a few lines of code, use the KNN algorithm in their projects for both prediction of continuous values and classification. Slightly more advanced users can employ such popular machine learning models as Linear Regression and Logistic Regression. These models are capable of processing large amounts of data and have a vast variety of applications.

There are numerous areas in which further work can be conducted. More supervised machine learning models can be implemented, but other types of machine learning can also be explored. The author of the thesis believes that the best way would be to study unsupervised machine learning, and specifically the countless number of neural networks that are gaining popularity every day.

Thus, the author of the thesis hopes that, thanks to his addition to the library, LightningChart JS users will be able to get acquainted with machine learning and begin to apply ML models in their activities.

REFERENCES

Adarsh Menon 2018. Linear Regression using Gradient Descent. https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931. 24.09.2022

Andrei D. Polyanin 2008. Partial differential equation. http://www.scholarpedia.org/article/Partial_differential_equation. 25.09.2022

Ashan Lakmal Thilakarathne 2020. https://medium.com/swlh/creating-a-model-for-weather-forecasting-using-linear-regression-b18c1590e8d7. Accessed 11.05.2023.

Aniruddha Bhandari 2020. Feature Scaling for Machine Learning: Understanding the Difference Between Normalization vs. Standardization. https://www.analyticsvidhya.com/blog/2020/04/feature-scaling- machine-learning-normalization-standardization/. Accessed 05.09.2022.

Ashish Mehta 2020. Why Is Logistic Regression Called "Regression" If It Is A Classification Algorithm? https://ai.plainenglish.io/why-is-logistic-regression-called-regression-if-it-is-a-classification-algorithm- 9c2a166e7b74. Accessed 23.09.2022.

Aurélien Géron 2017. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.

Clare Liu 2022. Data Transformation: Standardization vs Normalization. https://www.kdnuggets.com/2020/04/data-transformation-standardization-normalization.html. Accessed 05.09.2022.

Douglas C. Montgomery, Elizabeth A. Peck, G. Geoffrey Vining 2021. Introduction to Linear Regression Analysis, 6th Edition. Wiley.

Dhanoop Karunakaran 2020. Gradient descent algorithm explained with linear regression example. https://medium.com/intro-to-artificial-intelligence/gradient-descent-algorithm-explained-with-linear- regression-example-ff6b5491fdb9 Accessed 23.09.2022.

Felix Frohböse 2020. Machine Learning Case Study: Telco Customer Churn Prediction. https://towardsdatascience.com/machine-learning-case-study-telco-customer-churn-prediction-bc4be03c9e1d. Accessed 11.05.2023.

Natasha Sharma 2018. Spam Detection with Logistic Regression. https://towardsdatascience.com/spam-detection-with-logistic-regression-23e3709e522. Accessed 11.05.2023.

Hamza Zaman 2022. The Best Machine Learning Algorithm for Stock Prediction. https://hamza-zaman.medium.com/the-best-machine-learning-algorithm-for-stock-prediction-93fdce9e5b0e. Accessed 11.05.2023.

Himanshu Singh 2019. Everything you Need to Know About Hardware Requirements for Machine Learning. https://www.einfochips.com/blog/everything-you-need-to-know-about-hardware-requirements- for-machine-learning/. Accessed 07.08.2022.

Isha Salian 2018. SuperVize Me: What's the Difference Between Supervised, Unsupervised, Semi- Supervised and Reinforcement Learning? https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/. Accessed 06.06.2022.

Jason Brownlee 2019. A Gentle Introduction to Cross-Entropy for Machine Learning. https://machinelearningmastery.com/cross-entropy-for-machine-learning/. Accessed 14.09.2022.

Joe Godot 2023. https://ai.plainenglish.io/using-logistic-regression-to-predict-stock-movements-with-r-6375b35479bb. Accessed 11.05.2023.

Matthias Döring 2018. Supervised Learning: Model Popularity from Past to Present. https://www.kdnuggets.com/2018/12/supervised-learning-model-popularity-from-past-present.html. Accessed 10.07.2022.

Molly Ruby 2020. https://towardsdatascience.com/5-machine-learning-techniques-for-sales-forecasting-598e4984b109. Accessed 11.05.2023.

Moshe Binieli 2018. Machine learning: an introduction to mean squared error and regression lines. https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-c7dde9a26b93/. 24.09.2022

Matthew Bicknese 2020. Advantages and Basic Concepts Behind Tensorflow.js. https://medium.com/@mbicknese1/advantages-and-basic-concepts-behind-tensorflow-js-21eb1b9bb5d3. Accessed 30.06.2022.

Nikhil Adithyan 2020. Credit Card Fraud Detection With Machine Learning in Python. https://medium.com/codex/credit-card-fraud-detection-with-machine-learning-in-python-ac7281991d87. Accessed 11.05.2023.

Onel Harrison 2018. Machine Learning Basics with the K-Nearest Neighbors Algorithm. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm- 6a6e71d01761. Accessed 05.09.2022.

Pavel Romanov 2023. thesis-appendix. https://github.com/PavelRomanov2701/thesis-appendix Accessed 07.05.2022.

TensorFlow JS API. https://js.tensorflow.org/api/3.4.0/. Accessed 02.07.2022.

Vijay Kanade 2022. What Is Machine Learning? Definition, Types, Applications, and Trends for 2022. https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-ml/. Accessed 15.06.2022.

Vivek Chaudhary 2020. https://pub.towardsai.net/netflix-stock-prediction-model-a-comparative-study-of-linear-regression-k-nearest-neighbor-knn-4527ff17939b. Accessed 11.05.2023.