Trabalho Final
Inteligência Artificial

Professor: Gildo de Almeida Leonel
Aluno: Lucas Braga Fidelis

Relatório

- Informações sobre os dados coletados

Este relatório descreve o desenvolvimento de um modelo de aprendizado de máquina para prever preços de imóveis com base em seus metros quadrados, utilizando a biblioteca tensorflow.js. A aplicação carrega dados de um arquivo JSON, realiza a limpeza e normalização desses dados, treina um modelo sequencial de rede neural e visualiza os resultados.

1. carregar e limpar dados de vendas de imóveis.
2. criar e treinar um modelo de rede neural.
3. visualizar os resultados do treinamento.
4. testar o modelo com novos dados e comparar com os dados originais.

- Explicação sobre a preparação dos dados

1.A)O primeiro foi obtido os dados em crv e feita a conversão em JSON e limpar esses dados para garantir que apenas entradas válidas sejam utilizadas.

```
async function getData(){//obter e limpar dados
    const vcasaDataResponse = await
    fetch('vendacasa.json');
    const vcasaData = await vcasaDataResponse.json();
    const cleaned = vcasaData.map(vendacasa => ({
        sqft_living: vendacasa.sqft_living,
        price: vendacasa.price,
    })).filter(vendacasa => (vendacasa.sqft_living != null && vendacasa.price != null));

    return cleaned;
}
```

- Explicação sobre o problema a ser resolvido

1.B)Os dados são então visualizados utilizando ‘tfvis’ para criar um gráfico de dispersão que mostra a relação entre o preço e os metros quadrados do imóveis. Em seguida, os dados são convertidos para tensores e normalizados para o intervalo de 0 a 1.

```
async function run() {//carrega, visualiza e prepara os dados
    const data = await getData();
    const values = data.map(d => ({
        x: d.price,
        y: d.sqft_living
    }));

    tfvis.render.scatterplot(
        {name: 'Preço vs Metros quadrados de um imóvel'},
        {values},
        {
            xLabel: 'Preço',
            yLabel: 'Metros quadrados de um imóvel',
            height: 300
        }
    );
    const tensorData = convertToTensor(data);
        const { inputs, labels } = tensorData;

        await trainModel(model, inputs, labels);
        console.log("Treino Completo!");
        testModel(model, data, tensorData);
    }

document.addEventListener('DOMContentLoaded', run);
```

```
function convertToTensor(data){//converter dados para tensores

    return tf.tidy(() =>{
        //Passo 1. Embaralhe os dados
        tf.util.shuffle(data);

        // Etapa 2. Converta dados em tensor
        const inputs = data.map((d) => d.price);
        const labels = data.map((d) => d.sqft_living);

        const inputTensor = tf.tensor2d(inputs, [inputs.length, 1]);
        const labelTensor = tf.tensor2d(labels, [labels.length, 1]);
        //Etapa 3. Normalize os dados para o intervalo 0 -1 usando  escala min-max
        const inputMax = inputTensor.max();
        const inputMin = inputTensor.min();
        const labelMax = labelTensor.max();
        const labelMin = labelTensor.min();

        const normallizedInputs = inputTensor
        .sub(inputMin)
        .div(inputMax.sub(inputMin));

        const normallizedLabel = labelTensor
        .sub(labelMin)
        .div(labelMax.sub(labelMin));

    return{
        inputs: normallizedInputs,
        labels: normallizedLabel,
        inputMax,
        inputMin,
        labelMax,
        labelMin,
        };
    });
}
```

- Informações sobre o modelo de Machine Learning criado. (Quantas camadas, quais as funções de ativação e as medidas de avaliação utilizadas)

1.C) Um modelo sequencial é criado com três camadas densas: uma camada de entrada, uma camada intermediária com função de ativação ‘relu’, e uma camada de saída.

```
function createModel(){
    // Cria um modelo sequencial
    const model = tf.sequential();

    //Adiciona uma unica camada de entrada
    model.add(tf.layers.dense({inputShape: [1], units: 50, useBias:true}));

    model.add(tf.layers.dense({units: 50, activation: 'relu'}));

    //Adiciona uma camada de saida
    model.add(tf.layers.dense({units: 1, useBias: true}));

    return model;
}
```

1.D) O modelo é compilado com o otimizador ‘adam’ e a função de perda de erro quadrático médio. Em seguida, o modelo é treinado utilizando os dados normalizados.

```
async function trainModel(model,inputs,labels){
    //prepara o modelo para o treinamento.
    model.compile({
        optimizer: tf.train.adam(),//serve para ajustar os pesos(erros)
        loss: tf.losses.meanSquaredError,// serve de qual certo está errado nossa predição
        metrics: ["mse"],
    });
    const batchSize = 32;
    const epochs = 100;

    return await model.fit(inputs, labels, {
        batchSize,
        epochs,
        shuffle: true,
        callbacks: tfvis.show.fitCallbacks(
            { name: "Performance do treinamento"},
            ["loss", "mse"],
            { height: 200, callbacks: ["onEpochEnd"]}
        ),
    });
}
```

1.E) O modelo treinado é testado com novos dados e os resultados são comparados com os dados originais.

```
function testModel(model, inputData, normalizationData){
    const { inputMax, inputMin, labelMin, labelMax } = normalizationData;

    const [xs, preds ] = tf.tidy(() => {
        const xs = tf.linspace(0, 1, 100);
        const preds = model.predict(xs.reshape([100, 1]));

        const unNormXs = xs.mul(inputMax.sub(inputMin)).add(inputMin);

        const unNormPreds = preds.mul(labelMax.sub(labelMin)).add(labelMin);

        return [unNormXs.dataSync(), unNormPreds.dataSync()];
    });

    const predictedPoint = Array.from(xs).map((val,i) => {
        return { x: val, y:preds[i] };
    });

    const originalPoints = inputData.map((d) => ({
        x: d.price,
        y: d.sqft_living,
    }));

    tfvis.render.scatterplot(
        { name: "Previsões vs Dados originais"},
        {
            values: [originalPoints, predictedPoint],
            series: ["original", "predicted"],
        },
        {
            xLabel: "Preço",
            yLabel: "Metros quadrados de um imóvel",
            height: 300,
        }
    );
}
```

- Conclusão sobre o resultado encontrado após as previsões

Neste projeto, desenvolvemos e treinamos um modelo de aprendizado de máquina utilizando tensorflo para prever preços de imóveis com base em seus metros quadrados. Passamos por várias etapas. Além disso, investigamos como a complexidade do modelo afeta sua performance.

Comparei o erro médio quadrático MSE em dois cenários: um modelo com três camadas densas e outro modelo com quatro camadas densas.

O modelo original consistia em uma camada de entrada com 50 unidades, uma camada intermediária com 50 unidades e ativação ‘relu’, e uma camada de saída com uma unidade.

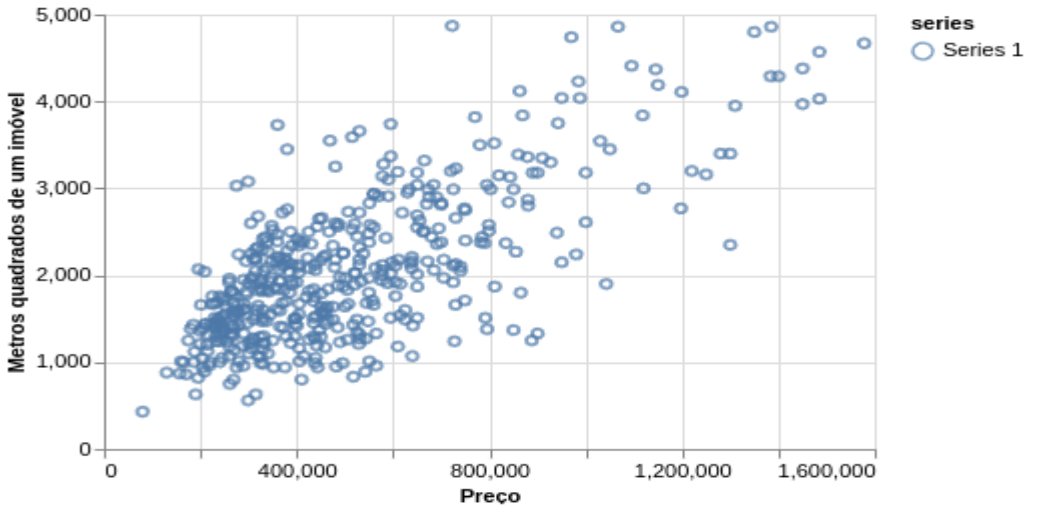
Para avaliar se aumentar a complexidade do modelo melhoraria seu desempenho, adicionamos uma camada densa extra com 50 unidades e ativação ‘relu’.

Conclusão foi que neste projeto, demonstramos que o modelo de aprendizado de máquina desenvolvido com tensorflow é eficaz para prever preços de imóveis com base nos metros quadrados. Adicionar mais complexidade ao modelo, como uma camada densa extra, não resultou em uma melhoria significativa no erro de predição. Portanto, um modelo com três camadas densas foi suficiente para alcançar um bom desempenho, equilibrando precisão e eficiência.

Modelo

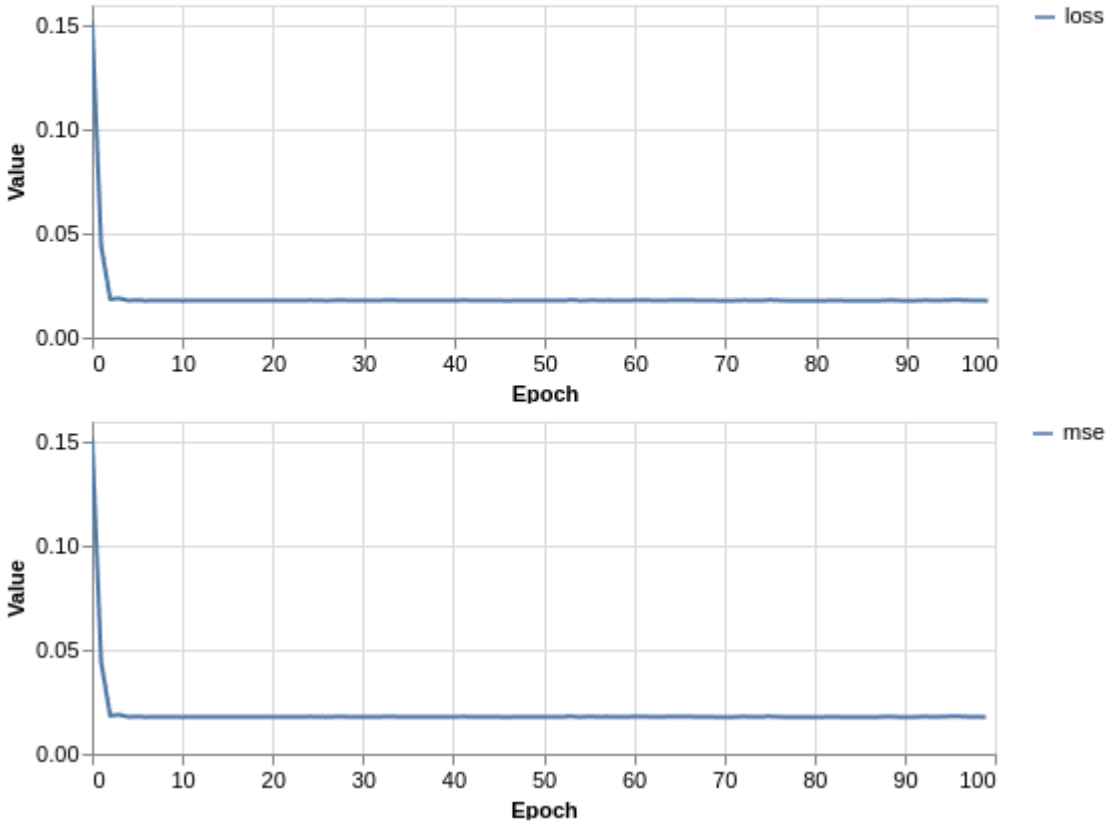
Layer Name	Output Shape	# Of Params	Trainable
dense_Dense1	[batch,50]	100	true
dense_Dense2	[batch,50]	2,550	true
dense_Dense3	[batch,1]	51	true

Preço vs Metros quadrados de um imóvel



Performance do treinamento

onEpochEnd



Previsões vs Dados originais