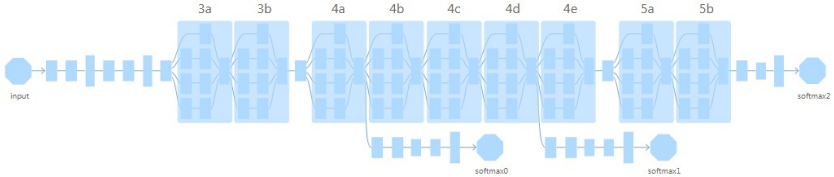
Google TensorFlow.js

浙江大学城市学院 ZHEJIANG UNIVERSITY CITY COLLEGE



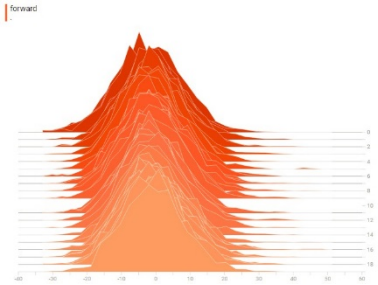
深度学习应用开发
基于TensorFlow的实践

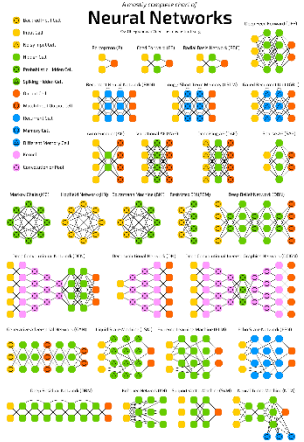
吴明晖 李卓蓉 金苍宏

浙江大学城市学院

计算机与计算科学学院

Dept. of Computer Science
Zhejiang University City College





浙江大学城市学院
ZHEJIANG UNIVERSITY CITY COLLEGE

基于TensorFlow.js鸢尾花品种识别

鸢尾花品种识别案例描述

鸢尾花卉数据集由Fisher在1936收集整理，别称Iris。数据集是一类多重变量分析的数据集。数据集包含150个数据集，分为3类，每类50个数据，每个数据包含4个属性。可通过花萼长度，花萼宽度，花瓣长度，花瓣宽度4个属性预测鸢尾花卉属于（Setosa-山鸢尾，Versicolour -杂色鸢尾，Virginica-维吉尼亚鸢尾）三个种类中的哪一类。

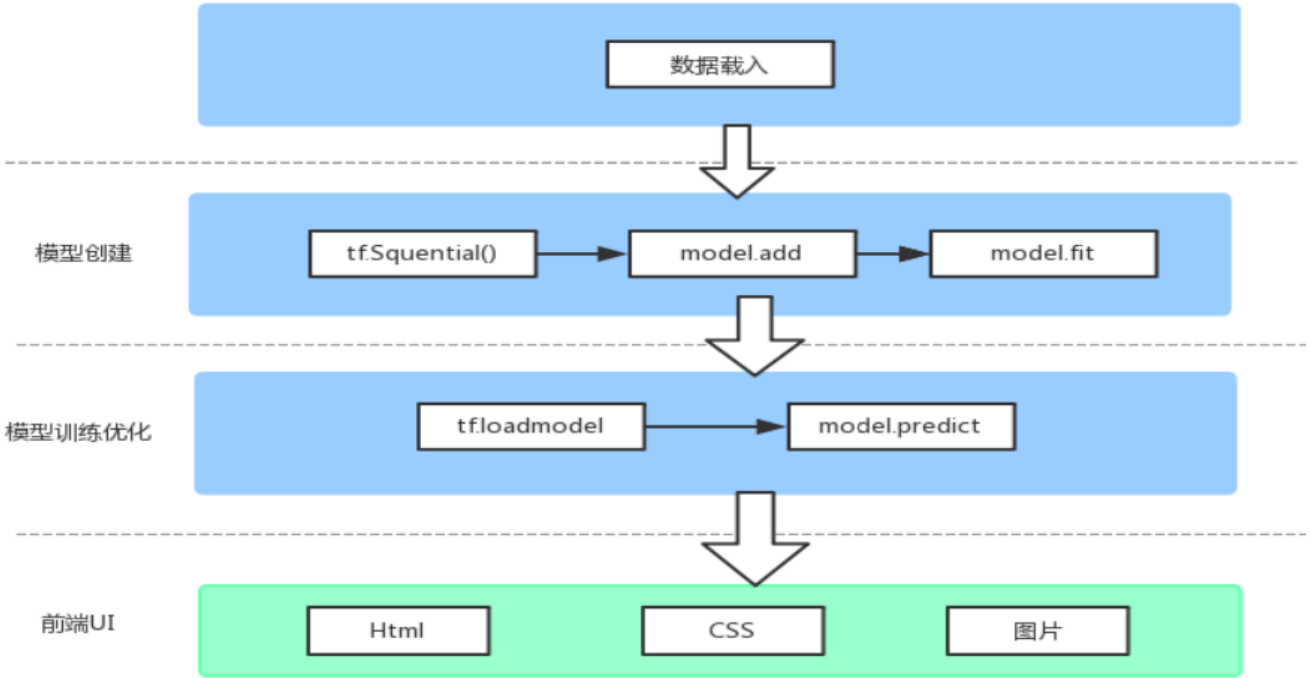
数据与模型

其数据格式显示为：

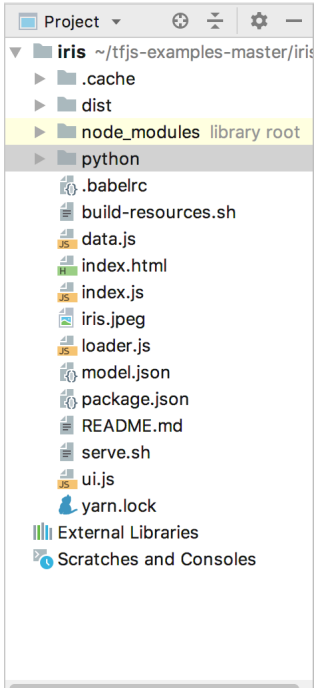
Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	*I. setosa*
4.9	3	1.4	0.2	*I. setosa*
4.7	3.2	1.3	0.2	*I. setosa*

Iris数据集分类可以使用逻辑回归Logistic Regression（LR）, K临近分析模型 K-Nearest Neighbors(KNN),分类回归树模型Classification and Regression Tree(CART), 朴素贝叶斯高斯模型Gaussian Naïve Bayes(NB)。

整体架构


数据载入
模型创建
tf.Squential()
model.add
model.fit
模型训练优化
tf.loadmodel
model.predict
前端UI
Html
CSS
图片

项目文件目录



鸢尾花分类案例

鸢尾属（拉丁学名：Iris L.），单子叶植物纲，百合目，鸢尾科多年生草本植物，有块茎或匍匐状根茎；叶剑形，嵌叠状；花美丽，状花序或圆锥花序；花被花瓣状，有一长或短的管，外弯，花柱分枝扩大，花瓣状而有颜色，外展而覆盖着雄蕊；子房下位，胚珠多数，果为蒴果。本属模式种：德国鸢尾（Iris germanica L.）原产欧洲，中国各地常见栽培。

训练模型

训练轮次： 40

学习率： 0.01

模型训练

模型保存和加载

Load hosted pretrained model

Load locally-saved model | Save model locally | Remove model locally | No locally-stored model is found.

Standing by.

Text Examples

花瓣长度	花瓣宽度	萼片长度	萼片宽度	真实分类	预测分类	概率
5.1 + -	3.5 + -	1.4 + -	0.2 + -			

数据生成

```
const IRIS_DATA = [
  [5.1, 3.5, 1.4, 0.2, 0], [4.9, 3.0, 1.4, 0.2, 0], [4.7, 3.2, 1.3, 0.2, 0],
  [4.6, 3.1, 1.5, 0.2, 0], [5.0, 3.6, 1.4, 0.2, 0], [5.4, 3.9, 1.7, 0.4, 0],
  [4.6, 3.4, 1.4, 0.3, 0], [5.0, 3.4, 1.5, 0.2, 0], [4.4, 2.9, 1.4, 0.2, 0],
  [4.9, 3.1, 1.5, 0.1, 0], [5.4, 3.7, 1.5, 0.2, 0], [4.8, 3.4, 1.6, 0.2, 0],
  [4.8, 3.0, 1.4, 0.1, 0], [4.3, 3.0, 1.1, 0.1, 0], [5.8, 4.0, 1.2, 0.2, 0],
  [5.7, 4.4, 1.5, 0.4, 0], [5.4, 3.9, 1.3, 0.4, 0], [5.1, 3.5, 1.4, 0.3, 0],
  [5.7, 3.8, 1.7, 0.3, 0], [5.1, 3.8, 1.5, 0.3, 0], [5.4, 3.4, 1.7, 0.2, 0],
  [5.1, 3.7, 1.5, 0.4, 0], [4.6, 3.6, 1.0, 0.2, 0], [5.1, 3.3, 1.7, 0.5, 0],
  [4.8, 3.4, 1.9, 0.2, 0], [5.0, 3.0, 1.6, 0.2, 0], [5.0, 3.4, 1.6, 0.4, 0],
  [5.2, 3.5, 1.5, 0.2, 0], [5.2, 3.4, 1.4, 0.2, 0], [4.7, 3.2, 1.6, 0.2, 0],
  [4.8, 3.1, 1.6, 0.2, 0], [5.4, 3.4, 1.5, 0.4, 0], [5.2, 4.1, 1.5, 0.1, 0],
  [5.5, 4.2, 1.4, 0.2, 0], [4.9, 3.1, 1.5, 0.1, 0], [5.0, 3.2, 1.2, 0.2, 0],
  [5.5, 3.5, 1.3, 0.2, 0], [4.9, 3.1, 1.5, 0.1, 0], [4.4, 3.0, 1.3, 0.2, 0],
  [5.1, 3.4, 1.5, 0.2, 0], [5.0, 3.5, 1.3, 0.3, 0], [4.5, 2.3, 1.3, 0.3, 0],
```

模型构建


```
// 模型定义，二层全连接
const model = tf.sequential();
model.add(tf.layers.dense(
    {units: 10, activation: 'sigmoid', inputShape: [xTrain.shape[1]]}));
model.add(tf.layers.dense({units: 3, activation: 'softmax'}));

const optimizer = tf.train.adam(params.learningRate);
model.compile({
  optimizer: optimizer,
  loss: 'categoricalCrossentropy',
  metrics: ['accuracy'],
});
```

模型训练


```
const lossValues = [];
const accuracyValues = [];
// Call `model.fit` to train the model.
const history = await model.fit(xTrain, yTrain, {
  epochs: params.epochs,
  validationData: [xTest, yTest],
  callbacks: {
    onEpochEnd: async (epoch, logs) => {
      // Plot the loss and accuracy values at the end of every training epoch.
      ui.plotLosses(lossValues, epoch, logs.loss, logs.val_loss);
      ui.plotAccuracies(accuracyValues, epoch, logs.acc, logs.val_acc);
    },
  }
});
```

预测模式

```
// Use a `tf.tidy` scope to make sure that WebGL memory allocated for the
// `predict` call is released at the end.
tf.tidy(() => {
  // Prepare input data as a 2D `tf.Tensor`.
  const inputData = ui.getManualInputData();
  const input = tf.tensor2d([inputData], [1, 4]);

  // Call `model.predict` to get the prediction output as probabilities for
  // the Iris flower categories.

  const predictOut = model.predict(input);
  const logits = Array.from(predictOut.dataSync());
  const winner = data.IRIS_CLASSES[predictOut.argMax(-1).dataSync()[0]];
  ui.setManualInputWinnerMessage(winner);
  ui.renderLogitsForManualInput(logits);
});
```

模型加载

```
export async function loadHostedPretrainedModel(url) {
  ui.status('Loading pretrained model from ' + url);
  try {
    const model = await tf.loadModel(url);
    ui.status('Done loading pretrained model.');
    return model;
  } catch (err) {
    console.error(err);
    ui.status('Loading pretrained model failed.');
  }
}
```

模型保存

```
保存目的地    方案字符串    代码示例
本地存储（浏览器）    localstorage://  await model.save('localstorage://my-model
IndexedDB（浏览器）   indexeddb://    await model.save('indexeddb://my-model-1'
触发文件下载（浏览器） downloads://    await model.save('downloads://my-model-1'
HTTP请求（浏览器） http:// 要么 https://  await model.save('http://model-server
文件系统（Node.js）    file://           await model.save('file:///tmp/my-model
```

不同浏览器数据储存方案：

Cookie 的大小不超过4KB，且每次请求都会发送回服务器；

LocalStorage 在 2.5MB 到 10MB 之间（各家浏览器不同），而且不提供搜索功能，不能建立自定义的索引。

IndexedDB 允许储存大量数据，提供查找接口，还能建立索引。IndexedDB 不属于关系型数据库（不支持 SQL 查询语句），更接近 NoSQL 数据库。