ML (theory and practice)

Author: Daria Shutina

23-09-08

Data representation

Continuous variables are those that can take any real-numbered value within a certain range (e.g. height, temperature, weight).

Binary variables are categorical variables that can take one of two possible values.

Embeddings are a technique used to represent categorical or discrete data (e.g. words, categories, IDs) in a continuous vector space.

Datasets

Datasets are structured collections of data used for training, testing, and evaluating models.

Datasets consist of **instances**. Each instance represents a single data entry. **Features** (or attributes) are characteristics or properties associated with each instance. **Labels** (or target values) represent the desired output that the model should learn to make for each corresponding instance.

The **training dataset** is used to train the ML model. It consists of a large number of labeled examples, allowing the model to learn the relationships between the features and labels.

The **validation dataset** is a separate portion of the data used during the training process to assess the model's performance and make decisions about hyperparameters or model selection. It helps prevent overfitting.

The **testing dataset** is used to evaluate the final performance of the trained model. It should be distinct from the training and validation datasets and provide an unbiased assessment of how well the model works with new data.

Train, validation, test

Hyperparameters are like settings for the ML algorithm. At the beginning, hyperparam values are set based on what you believe would result in a good performance. This can be based on research or experience. After the hyperparams are set and the data is ready, we **train** the model on the training dataset.

The model's performance depends on the dataset it is being trained on. One of the main aims is to make the model robust, meaning the model gives consistent results that are correct most of the time.

After the training is done, results of the model might be not good enough, because hyperparameters do not have an optimal value. On the next step -- **tuning** -- different combinations of hyperparams are tested.

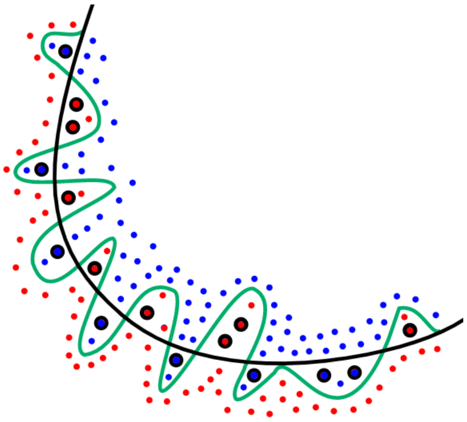
So, the main steps of preparing the model:

- **train**: find the model params. The model is trained with the training dataset.
- **validation**: tune hyperparameters. We tune hyperparams and train again, also checking for the performance. If needed, the tuning can be done again. This way the model is being optimized to show the best performance on the validation data.
- **test**: run the model on data the model has never seen.

We need to split data between train, validation and test datasets. It can be done randomly or sequentially.

Overfitting

In simple words, the model "overfits" when it is correct on every instance of the training dataset but has poor fit with new, unknown datasets.



Here, blue and red dots represent different classes, and the model is learning to distinguish between them. The green line is an overfitted curve, which means the model gives a correct answer on every instance from the training dataset. On the other hand, the black line is more general.

As a result, the model with the black curve has a higher probability of being correct on a new data, since the green curve is too fluctuating.

Loss functions

A **loss function** (or cost function) quantifies the difference between the predicted values generated by a model and the target values in the training dataset.

The goal in training a model is to minimize this loss function. The aim is to find the model's parameters that result in the lowest possible loss.

Example

We want know whether the user clicks the video. Lets use a **log loss function**:

$$L = -\log p \cdot y + \log(1-p) \cdot (1-y)$$

Here p is a prediction, it is calculated by our model, and y is a label (1 if click, 0 otherwise).

- $L \to \min$
- $y = 1 \ \Rightarrow \ -\log p \to \min$
    - $-\log 0 = +\infty$
    - $-\log 1 = 0$
- $y = 0 \ \Rightarrow \ \log(1-p) \to \min$
    - $\log(1-0) = 0$
    - $\log(1-1) = +\infty$

23-09-15

Linear models

Jupiter notebook practice

Moore-Penrose inverse method

We want to solve the equation $Xa = y$ => $a = X^{-1}y$. If X is not inverse (not squared), use $a = (X^T X)^{-1} X^T y$.

```
def solve_pseudo_inverse(X, y):
    matrix = np.linalg.inv(np.dot(X.T, X))
    return np.dot(np.dot(matrix, X.T), y)

print(solve_pseudo_inverse(X_train, y_train))
```

Stochastic Gradient Descent function

We have n features $(x_0, \ldots, x_{n-1})$ and want to find coefficients $a_0, \ldots, a_{n-1}$:

$$y_{pred} = f(A, X) = a_0 x_0 + \ldots + a_{n-1} x_{n-1}$$

The loss function will be the distance from real and predicted results:

$$L = (y_{pred} - y_{real})^2 = (\sum_0^{n-1} a_i x_i - y_{real})^2$$

$$L \to \min$$

We are going to use the gradient of the loss function to find coefficients. The gradient is a matrix consisting of derivatives, each of them has a form $\frac{\partial L}{\partial a_i} = L'_{a_i} = 2(a_0 x_0 + \ldots + a_{n-1} x_{n-1} - y_{real}) x_i$.

```
def solve_sgd(X, y, iterations, learning_rate):
    n = X.shape[1]
    a = np.zeros(n)

    for i in range(iterations):
        t = 2 * (np.dot(X, a) - y) * X.T
        t = np.mean(t, axis = 1)
        a -= t * learning_rate
    return a

print (solve_sgd(X_train, y_train, 1000, 0.01))
```

The `solve_sgd` function first initializes vector `a` with zeros and iteratively updates it in order to minimize the loss function error. In each iteration, it computes the gradient `t` for the current vector `a` and updates the coefficients.

`np.mean(t, axis = 1)` finds the average over all examples in the dataset. For example, t_1 is the average of derivatives' values for the coefficient a_1.

`learning_rate` is a hyperparameter that determines the step size in the direction of the gradient. Typically, it is a positive number between 0 and 1 that scales the gradient before applying it to update coefficients.

Decision trees

Parameters used in a decision tree are features, thresholds and conditions. Every node contains a condition - question of a form like `if feature1 < threshold1`.

The implementation for decision trees is `catboost`.

Boosting forest

$y = tree_0(x) + \ldots + tree_{n-1}(x)$.

Trees are not deep (3 to 6 levels). Each tree makes a tiny step towards the goal. When trees from 0 to $i-1$ are known, the i-th tree is found from the equation $y - tree_0(x) - \ldots - tree_{i-1}(x) = tree_i(x)$.

Unlike decision trees, boosted trees do not overfit quickly, since a tree with thousands of params is more likely to overfit.

```python
import catboost


def solve_catboost(X, y, iterations, learning_rate, depth):
    model = catboost.CatBoostRegressor(iterations = iterations,
                                       learning_rate = learning_rate,
                                       depth = depth)
    model.fit(X, y, verbose = False)
    return model


X_train, y_train = generate_data([1, 2, 3], 1000, 0.01)
X_test, y_test = generate_data([1, 2, 3], 100, 0.01)
model = solve_catboost(X_train, y_train, 1000, 0.1, 6)
```

A `CatBoostRegressor` model is trained using the `solve_catboost` function:

- `iterations` is the amount of decision trees

- `learning_rate` is used for controlling the step size in the direction of the gradient
- `depth` - depth of decision trees

Random forest

$$y = avg(tree_0(x), tree_1(x), \ldots, tree_{n-1}(x))$$

Each tree is trained on a sample of the original dataset.

To avoid overfitting, a technique called *bagging* is used:

- sample bagging: multiple decision trees are trained on random subsets (repeating data points is allowed) of the training data
- feature bagging: a random subset of features is selected for each tree

23-09-22

Language models

Language models are designed to understand and generate human language text. They are used for:

- classification
    - token classification is the task of assigning a label or category to each individual token or word in a given text sequence. Each token is typically classified independently based on its context within the sequence.
    - Sequence classification involves assigning a single label or category to an entire sequence of tokens or words.
- retrieval (you have a set of tokens and want to extract some data related to the given tokens)
- translation
- generation
    - Q&A: given a question and an article, come up with an answer based on the article
    - summarization: summarize data from the article
    - chat: e.g. ChatGPT

Word embeddings

Imagine we have a document, each word from the document has a form $[0, \ldots, 0, 1, 0, \ldots, 0]$ as a vector. Depending on the document size and content, the size and amount of vectors can be huge.

Embeddings are vector representations of words in a continuous vector space. The idea of using embeddings is to transform the word's vector $[0, \ldots, 0, 1, 0, \ldots, 0]$ into a n-dimensional vector $[e_1, \ldots, e_n]$.

Skip-gram and Continuous Bag of Words (CBOW) are two popular algorithms used for training word embeddings. Skip-gram is often better at capturing semantic[1] relationships and is more suitable for tasks like word analogy (e.g. "king - man + woman = queen"). CBOW is faster to train and can perform well on syntactic[2] tasks.

Skip gram

The main goal of the Skip-gram model is to predict the context words (surrounding words) given a target word. The input is a target word, and the output is a probability distribution over all the words in the vocabulary.

Example: given the sentence "I love to eat pizza," if the target word is "love," Skip-gram aims to predict the surrounding words like "I," "to," "eat," and "pizza."

CBOW

CBOW tries to predict the target word given a context (surrounding words). The input is a context (a set of surrounding words), and the output is the target word. The model attempts to maximize the probability of the target word given the context words.

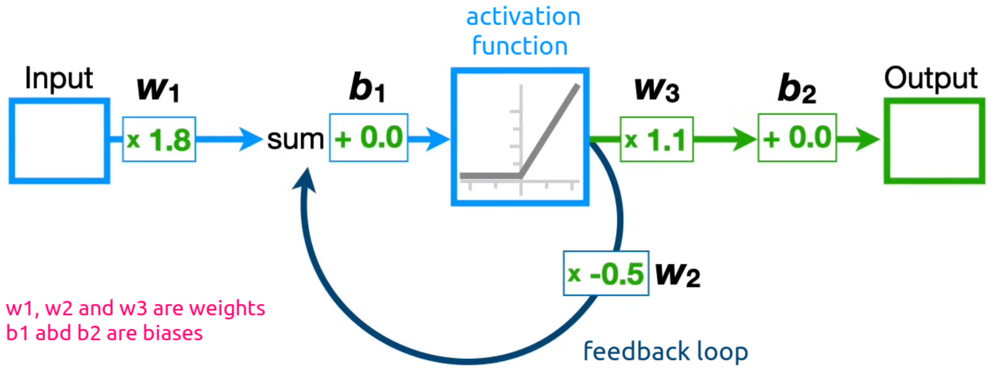
Example: given the same sentence "I love to eat pizza," if the context is "I, to, eat, pizza," CBOW aims to predict the target word "love."

Markov chain

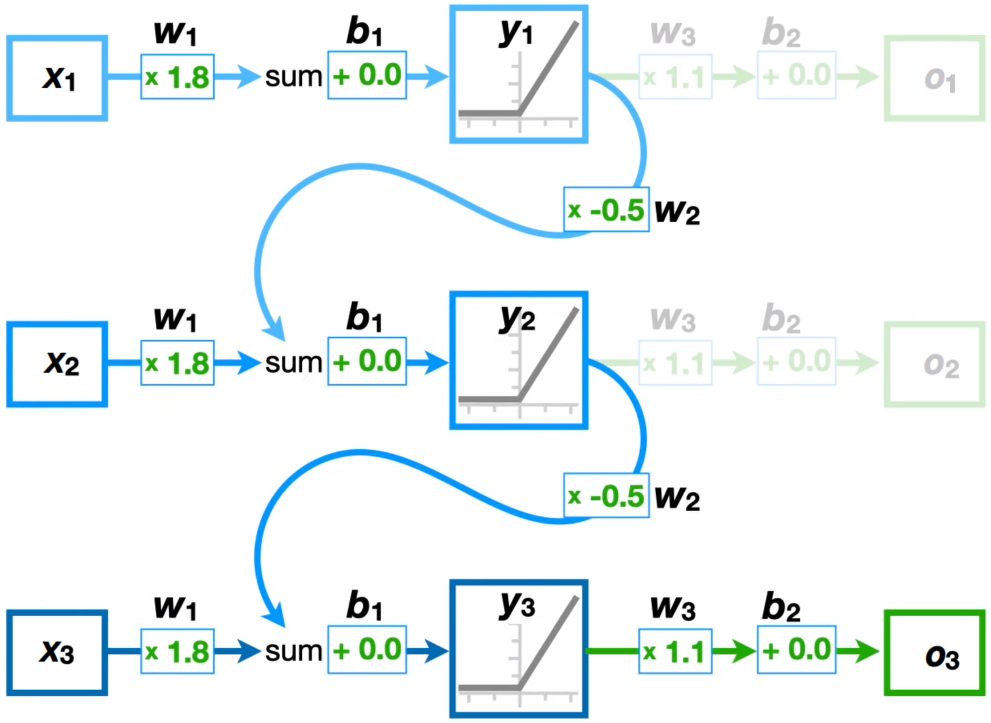
When constructing a phrase, there is a probability for each next word. The probability depends on the previous knowledge.

RNNs

Recurrent neural networks deal with an arbitrary amount of input values. Like any other networks, RNNs have weights, biases, layers and activation functions. The difference is they also have **feedback loops**. The feedback loop makes it possible to use sequential input values to make predictions.



To understand how the feedback loop works, we can **unroll** it. Thus, we end up with a new neural network that has several inputs $x_1, \dots, x_n$ and several outputs $o_1, \dots, o_n$. The same weights and biases are shared across every input.



If we need the output o_n, intermediate results are simply ignored.

The problem with RNNs is that the more we unroll it, the harder it is to train. The problem is called **The Vanishing/Exploding Gradient Problem**.

When the gradient contains a huge number, relatively large steps are taken in the Gradient Descent algorithm. Thus, instead of finding the optimal parameter, we will just bounce around it. It explains the Exploding Gradient Problem.

One way to prevent The Exploding Gradient problem is to limit the parameter w_2 to values < 1. We end up taking steps that are too small and hit the maximum number of steps we are allowed to take before finding the optimal value. It explains the Vanishing Gradient Problem.

Transformers

Transformers represent a newer and more specialized type of neural networks, well-suited for tasks involving sequences of data, such as text, time series, and more. They can process input data in parallel, unlike RNNs, which are sequential.

Positional Encoding is a technique transformers use to keep track of word order.

[1] Semantic refers to the meaning of words. It deals with the interpretation of words and their relationship to each other. Example: In the sentence "The cat chased the mouse," the semantic meaning is that the cat is pursuing the mouse, indicating a specific relationship between the two animals. ↩

[2] Syntactic refers to the structure and arrangement of words in sentences and how they form grammatically correct sentences. Example: In the sentence "The cat chased the mouse," the syntactic structure follows the typical subject-verb-object order in English. ↩