

Homework 4 - Machine Learning Training Pipeline - Spring 2025

Essay Question: *Machine learning (ML) pipelines offer many benefits to the ML engineer responsible for designing, training, and evaluating ML models that perform well without overfitting the available data. These benefits are ...* List what you see as the benefits of these pipelines, for each benefit include a sentence describing how that benefit helps the process of model development. Conclude with a sentence summarizing the critical role that such pipelines play in developing ML applications.

Notebook Assignment 1 (Naive Bayes Baseline Model) :

Introduction: A widely used baseline model for text classification is the Naive Bayes model. The text classification problem has input samples $\mathbf{x} \in \{0, 1\}^d$ that are binary vectors of dimension d . The i th component, x_i , represents the i th word in a predefined vocabulary list and $x_i = 1$ if the i th word in the vocabulary is present in a text document and is 0 otherwise. The target, $y \in \{0, 1\}$, for input sample \mathbf{x} is 1 if the text document is in class 1 (let's say a paper about deep learning) otherwise it is 0 (a document that is about something other than deep learning). The text classification problem uses the dataset to train a model that predicts which class a document belongs to. In general, we would classify the document, \mathbf{x} , with the class that has the highest posterior probability $\Pr(y | \mathbf{x})$. We will use our Naive Bayes' baseline model to compute an estimate, $\widehat{\Pr}(y | \mathbf{x})$, of that posterior probability, which we would then use to compute the baseline accuracy levels we need our deep learning model to beat.

Our estimate for $\widehat{\Pr}(y | \mathbf{x})$ is obtained by first using Bayes' theorem to note that

$$\Pr(y | \mathbf{x}) = \frac{\Pr(\mathbf{x} | y)\Pr(y)}{\Pr(\mathbf{x})} \propto \Pr(\mathbf{x} | y)\Pr(y)$$

The probability $\Pr(y)$ can be estimated from the dataset by simply counting up how many samples are in class $y = 1$ or class $y = 0$ and dividing by the total number of samples in the dataset.

We need to estimate the a priori probability $\Pr(\mathbf{x} | y)$, which is what our "trained" model should do. But to get a crude baseline model for this, we can simply assume that \mathbf{x} is a random variable whose components, x_i , are independent Bernoulli random variables

$$\Pr(x_i | y) = x_i \Pr(x_i = 1 | y) + (1 - x_i) \Pr(x_i = 0 | y)$$

The baseline model's estimate of the document's a priori probability is therefore,

$$\Pr(\mathbf{x} | y) = \prod_{i=1}^d (x_i \Pr(x_i = 1 | y) + (1 - x_i) \Pr(x_i = 0 | y))$$

The probability $\Pr(x_i = 1 | y)$ can be estimated from the dataset by simply counting up how many times word i appears in documents that are of class y .

So we can now propose the following procedure to generate a Naive Bayes' baseline model for the text classification problem

1. Define vocabulary list V where the cardinality of V determines the dimension of the input vectors, \mathbf{x} .

2. Count the following in the dataset

- N = total number of documents in the dataset
- N_m = number of documents labelled with class m for $m = 1, 2, \dots, M$
- $n_m(x_i)$ = number of documents of class k containing word i

3. Estimate the likelihoods $\widehat{\Pr}(x_i | y = m) = \frac{n_m(x_i)}{N_m}$

4. Estimate the priors $\widehat{\Pr}(y = m) = \frac{N_m}{N}$

5. Estimate the baseline model's prediction of the posterior *log likelihood* for each class

$$\begin{aligned}\log(\widehat{\Pr}(y = m | \mathbf{x})) &= \log(\widehat{\Pr}(\mathbf{x} | y = m)) + \log(\Pr(y = m)) \\ &= \log(\widehat{\Pr}(y = m)) + \sum_{i=1}^{|V|} \log \left(x_i \widehat{\Pr}(x_i | y = m) + (1 - x_i)(1 - \widehat{\Pr}(x_i | y = m)) \right)\end{aligned}$$

6. The baseline model's prediction is to select the class that has the largest log likelihood. One then evaluates the model's performance (accuracy or confusion matrix) on the test data.

For this assignment you'll use the preceding method to compute a baseline model for the IMDB Movie Review dataset and then compare that baseline against a sequential model trained on the same dataset. The tasks that need to be completed for this exercise are enumerated below.

1. **TO DO:** Load the IMDB dataset from tensorflow and create the train/test input/target sample arrays assuming a 10000 word vocabulary. Print the shape of the train/test input/target arrays and print the first sample's input/label.
2. **TO DO:** Write a function that encodes each input sample as a one-hot encoded vector whose k th component is 1 if that k th vocabulary word appears in the review and is zero otherwise. Use your function to encode the training/testing inputs as one-hot encoded vectors, and re-type each input as float32. Print out the shape of training input data. Print out the number of vocabulary words in the first review and print out the word indices.
3. **TO DO:** Write a function (`baseline_model`) that computes the statistics $\Pr(y = m)$ and $\Pr(x_k | y = m)$ where m is the class label in $\{0, 1\}$. Your function should take the training dataset and returns the four probabilities in a list called `model`. The first two components are the probabilities $\Pr(y = 0)$ and $\Pr(y = 1)$. The last two components are arrays containing the probabilities $\Pr(x_k | y = 0)$ and $\Pr(x_k | y = 1)$. Print the class probabilities $\Pr(y = 0)$ and $\Pr(y = 1)$. Verify the shape of the conditional sample probabilities.
4. **TO DO:** Write a function that evaluates the accuracy of the baseline model's predictions on the testing data. In writing your function use the sum of log likelihoods, rather than product of likelihoods. This turns out to be more robust computationally. Print the accuracy returned by your evaluation function.

5. **TO DO:** Use the training/testing data arrays to form tf dataset objects `train_ds` and `test_ds`. Assume a batch size of 256 and print out the shape of each dataset object along with the number of batches in each dataset. Take the training dataset object (`train_ds`) and use an 80/20 split to partition it into a p-training dataset and a validation dataset object. Print the number of p-training batches and validation batches in your dataset objects.
6. **TO DO:** Instantiate a sequential model whose input layer takes the 10000 dimensional input sample (1-hot encoded) and maps it through two dense layers of 4 nodes, each using relu activation. The output layer of the model is a single node with sigmoid activation. Compile your model using an RMSprop optimizer with a learning rate of 10^{-4} , binary crossentropy loss, and "accuracy" as the metric. Print the model summary and determine the number of trainable weights.
7. **TO DO:** Train your model for 60 epochs using the p-training data and validation dataset objects. Create a callback that saves the "best model" with the smallest validation loss. Have your fit routine return the history object.
8. **TO DO:** Evaluate the best model's test accuracy. Print out the model's test accuracy alongside the baseline accuracy you computed above. Compare the two. Plot the training curves for this history object.

Notebook Assignment 2 (Hyperband Tuner): One of the most time-consuming tasks in deep learning is hyperparameter tuning. These hyper-parameters include the number of layers and number of nodes in a deep network. It may include optimizer parameters such as the learning rate or momentum term. It may also include parameters used by a dropout layer or L_2 regularization. Hyperparameter tuning is usually done by creating a finite grid of model configurations, evaluating the loss or accuracy achieved by training a model with the given hyper-parameters, and then using the outcomes to select a new set of parameterized model configurations. The selection criterion is chosen to "reduce" the number of configurations until the recursive application of this base process identifies a single parameter set that trains a model with the smallest loss (highest accuracy).

Two methods for hyperparameter tuning are Bayesian optimization and the Hyperband algorithm. Both methods have been implemented in the KerasTuner tool which is available in TensorFlow. This notebook assignment has you code a simple tuning algorithm whose basic recursion is the foundation of the Hyperband algorithm. You will be working with the fashion MNIST classification problem. The tasks for this exercise are enumerated below.

1. Use `set_tf_loglevel` to reduce the number of logging messages. This function is in `HW4utils.py` and should be called as

```
set_tf_loglevel(logging.FATAL)
```

TO DO: Use the other `HW4utils` functions to load the fashion MNIST dataset and create a p-training, validation, and test dataset. Use a 2/3 split in splitting the training data into p-training and validation data.

2. **TO DO:** Create a function `model_builder` that takes a hyperparameter (`hp`) representing the number of nodes in a MLP's hidden layer. Your model builder function will build a model that takes an input with shape (784,), has "hp" nodes in the first hidden dense layer with an `relu` activation, and has 10 nodes in the final output layer with a `softmax` activation. Your function should also compile the model using an `rmsprop` optimizer, sparse categorical crossentropy loss, and "accuracy" metric. The function should return the handle of the instantiated model.

TO DO: You will also need to create another function that takes a set (`hp_bag`) of hyperparameters and builds a model for each hyperparameter in the bag. For each of these models, train the model for 0 epochs to complete the build (set `verbose=0` to reduce the output). You will then need to save the model as

```
"model"+str(hp)+".keras"
```

so it can be used later.

3. This is the main part of the exercise. You will implement a simplified version of the hyperband algorithm

Consider a two layer dense sequential model. The first layer has `hp` nodes with `RELU` activation, followed by a 10 node dense layer. We will treat the number of nodes in that first dense layer as the hyperparameter. Load the `fashion-mnist` dataset and create a numpy array, `hp_bag` which is the set of hyperparameters we want to test.

```
import numpy as np
hp_bag = np.arange(16, 320, 32)
```

`hp_bag` is therefore a collection of `n_hidden` hyper-parameters that we search for the "best" model using the Hyperband algorithm [1]. This task has you implement a simplified form of the hyperband algorithm.

The hyperband algorithm [1] is based on the idea that you start by training a collection (bag) of models for a few epochs, say 5, and then discard from the bag those models that performed worst. You then continue training for another 5 epochs, discard the worst ones, and continue until there is only one model left, which you then train to completion.

TO DO: Use this algorithm to train a simple MLP model for the Fashion-MNIST dataset. Your earlier `model_builder` function constructs models with two dense layers in which the number of nodes in the hidden layer is the hyperparameter you are tuning. So you will need to create a numpy array, `hp_bag`, containing the hyperparameters we want to test. In this case, select an array that goes from 16 to 320 in increments of 32. Then write a script that trains the configurations in `hp_bag` for just 5 epochs on the p-training data. Evaluate the loss achieved by all of the trained models on the validation data and remove at most (`num_remove`) 3 configurations with the largest losses from `hp_bag`. Your algorithm will repeat this process until there is only one model left in the bag.

4. **TO DO:** To finish the exercise, train the remaining model for 30 epochs, generate the training curves, and determine the best model's test accuracy.

NOTE: TensorFlow has an automated tuning function based on the Hyperband algorithm described above [2]. TensorFlow's implementation of the algorithm is much more efficient than the script you probably wrote and it is useful to know how to use it. Tutorial can be found on tensorflow website. Feel free to explore as it may be useful for those of you considering doing a project.

References

- [1] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.
- [2] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Kerastuner. <https://github.com/keras-team/keras-tuner>, 2019.