Model Tuning Interview Questions

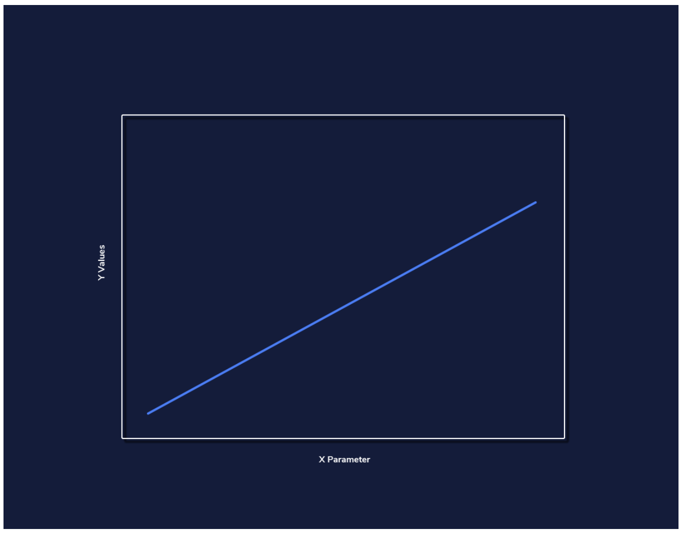
Bagging at Random Forest

Trees in a random forest classifier are created by using a random subset of the original dataset with replacement. This process is known as bagging. Bagging prevents overfitting, given that each individual tree is trained on a subset of original data.

Loss Functions

After data has been sent through a neural network, the error between the predicted values and actual values of the training data is calculated using a *loss function*. There are two commonly used loss calculation formulas:

- Mean squared error which is used for regression problems — the gif above shows how mean squared error is calculated for a line of best fit in linear regression.
- Cross-entropy loss, which is used for classification learning models rather than regression.



Train and Test Sets

When training a deep learning model (or any other machine learning model), split your data into *train* and *test* sets. The train set is used during the learning process, while the test set is used to evaluate the results of your model.
To perform this in Python, we use the `train_test_split()` method from the scikit-learn library.

```
from sklearn.model_selection import
train_test_split

# Here we chose the test size to be 33%
of the total data, and random state
controls the shuffling applied to the
data before applying the split.

features_train, features_test,
labels_train, labels_test =
train_test_split(features, labels,
test_size=0.33, random_state=42)
```

Grid Search for Deep Learning

When tuning a deep learning model, one can use *grid search*, also called *exhaustive search*, to try every combination of desired hyperparameter values.
If, for example, we want to try learning rates of 0.01 and 0.001 and batch sizes of 10, 30, and 50, grid search will try six combinations of parameters (0.01 and 10, 0.01 and 30, 0.01 and 50, 0.001 and 10, and so on).
To implement this in Python, we use `GridSearchCV` from scikit-learn. For regression, we need to first wrap our neural network model into a `KerasRegressor`. Then, we need to setup the desired hyperparameters grid (we don't use many values for the sake of speed). Finally, we initialize a `GridSearchCV` object and fit our model to the data. The implementation of this is shown in the code snippet.

```
model =
KerasRegressor(build_fn=design_model)

# batch sizes and epochs to test
batch_size = [4, 8, 16, 64]
epochs = [10, 50, 100, 200]
# setting up our grid of parameters
param_grid = dict(batch_size=batch_size,
epochs=epochs)

# initiliazing a grid search
grid = GridSearchCV(estimator = model,
param_grid=param_grid, scoring =
make_scorer(mean_squared_error,
greater_is_better=False))
# fitting the results
grid_result = grid.fit(features_train,
labels_train, verbose = 0)
```

Print Share