


IIT Madras

ONLINE DEGREE

Machine Learning Practice
Online Degree Programme
B. Sc in Programming and Data Science
Diploma Level
Dr. Ashish Tendulkar
Indian Institute of Technology – Madras

Finetuning ML models

(Refer Slide Time: 00:10)



Step 6: Finetune your model

- Usually there are a number of hyperparameters in the model, which are set manually.
- Tuning these hyperparameters lead to better accuracy of ML models.
- Finding the best combination of hyperparameters is a search problem in the space of hyperparameters, which is huge.

Grid search

- Scikit-Learn provides a class GridSearchCV that helps us in this pursuit.

```
1 from sklearn.model_selection import GridSearchCV
```

- We need to specify a list of hyperparameters along with the range of values to try.
- It automatically evaluates all possible combinations of hyperparameter values using cross-validation.

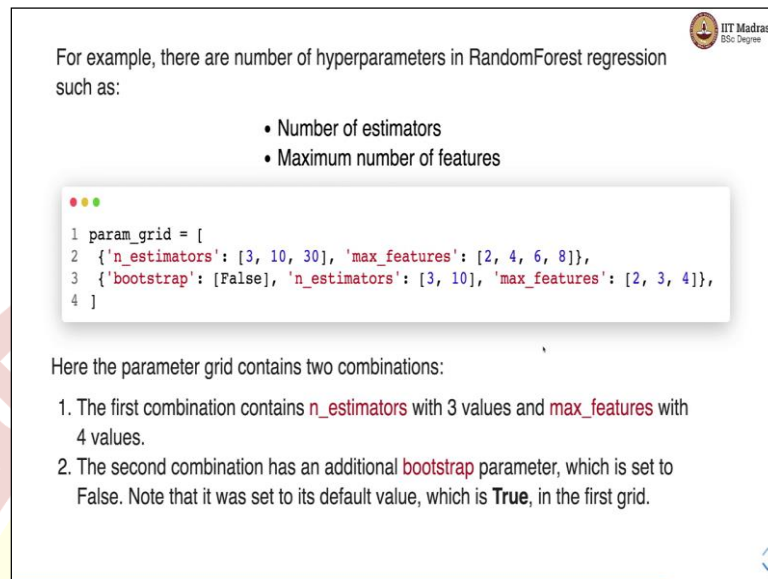
Namaste welcome to the next video of machine learning practice course. In this video, we will discuss the remaining steps of the end-to-end machine learning project. We will begin by discussing the step that involves fine-tuning the machine learning model. Usually, there are a number of hyper parameters in machine learning models which are set manually by the machine learning model developer.

Tuning these hyper parameters usually lead to better accuracy of machine learning models. However, finding the best combination of hyper parameters in a brute force manner is a challenging task. The hyper parameter space is huge and finding the best combination is a search problem in this huge parameter space. Fortunately, we have couple of ways in which we can perform this activity in scikit-learn package.

The first one is the grid search. We have a class GridSearchCV that helps us in finding the best combination of hyperparameters. We can import the GridSearchCV from the model selection package. We need to specify a list of hyperparameters along with the range of

values to try. Once we specify these values GridSearchCV would automatically evaluate all possible combinations of hyper parameters using cross validation.

(Refer Slide Time: 01:42)



For example, there are number of hyperparameters in RandomForest regression such as:

- Number of estimators
- Maximum number of features

```
1 param_grid = [  
2   {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},  
3   {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},  
4 ]
```

Here the parameter grid contains two combinations:

1. The first combination contains `n_estimators` with 3 values and `max_features` with 4 values.
2. The second combination has an additional `bootstrap` parameter, which is set to False. Note that it was set to its default value, which is `True`, in the first grid.

Let us take a concrete example of performing a hyperparameter search in a random forest regression problem. There is a number of hyperparameters in random forest regression such as a number of estimator's maximum number of features. So, we specify the parameter grid or the combinations of parameters to try in form of a parameter grid. In this case, we specify number of estimators to try to be a list of three numbers 3 ,10 and 30 and the maximum features that we want to try are 2, 4, 6, and 8.

We also specify the value of bootstrap to be false and the number of the estimator to be 3 and 10 and maximum features to be 2, 3, and 4. So, we specify here two sets of hyperparameter search problems. There are two combinations. the first combination contains a number of estimators with three values and maximum features with four values and the second combination has an additional bootstrap parameter which is set to false.

(Refer Slide Time: 03:01)

Let's compute the total combinations evaluated here:

1. The first one results in $3 \times 4 = 12$ combinations.
2. The second one has 2 values of `n_estimators` and 3 values of `max_features`, thus resulting $2 \times 3 = 6$ in total of values.

The total number of combinations evaluated by the parameter grid $12 + 6 = 18$

Let's create an object of `GridSearchCV`:

```
1 grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
2                           scoring='neg_mean_squared_error',
3                           return_train_score=True)
```

Let us compute the total number of combinations that will be evaluated here. The first one has three values for a number of estimators and four values for a maximum number of parameters that would lead to 12 combinations and the second one has two values of a number of estimators three values of maximum features thus resulting in 6 combinations. So, the total number of combinations that will be evaluated by the parameter grid is the sum of these two combinations which is 12 plus 6 leading to 18 combinations.

Let us create a `GridSearchCV` object with the parameter grid in `GridSearchCV`. We specify the estimator which is forest underscore reg this is the random forest regressor object the parameter grade number of cross-validation fold, the scoring scheme and a flag that specifies whether we want to also return the training scores.

(Refer Slide Time: 04:09)

Let's create an object of `GridSearchCV`:

```
1 grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
2                           scoring='neg_mean_squared_error',
3                           return_train_score=True)
```

- In this case, we set **cv=5** i.e. using 5 fold cross validation for training the model.
- We need to train the model for 18 parameter combinations and each combination would be trained 5 times as we are using cross-validation here.
- The total model training runs = $18 \times 5 = 90$

So, we need to train this model for 18 parameter combinations and each combination would be trained five times because the number of cross-validation is set to 5. So, in all will be performing 90 model training runs.

(Refer Slide Time: 04:38)

Let's launch the hyperparameter search:

```
1 grid_search.fit(wine_features_tr, wine_labels)

GridSearchCV(cv=5, error_score=nan,
              estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                                criterion='mse', max_depth=None,
                                                max_features='auto',
                                                max_leaf_nodes=None,
                                                max_samples=None,
                                                min_impurity_decrease=0.0,
                                                min_impurity_split=None,
                                                min_samples_leaf=1,
                                                min_samples_split=2,
                                                min_weight_fraction_leaf=0.0,
                                                n_estimators=100, n_jobs=None,
                                                oob_score=False, random_state=None,
                                                verbose=0, warm_start=False),
              iid='deprecated', n_jobs=None,
              param_grid=[{'max_features': [2, 4, 6, 8],
                           'n_estimators': [3, 10, 30]},
                           {'bootstrap': [False], 'max_features': [2, 3, 4],
                           'n_estimators': [3, 10]}],
              pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
              scoring='neg_mean_squared_error', verbose=0)
```

So, we can launch the hyper parameter search by calling fit function on the GridSearchCV.

(Refer Slide Time: 04:53)

The best parameter combination can be obtained as follows:

```
1 grid_search.best_params_


{'max_features': 6, 'n_estimators': 30}
```

Let's find out the error at different parameter settings:

```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(-mean_score, params)
```

Once the fit is complete we get the best combination of hyper parameter with the best underscore parameter underscores member variable of the object. Here the best combination of parameters seems to be maximum features equal to 6 and the number of estimators equal to 30. Let us find out errors at different parameter settings we can find it out with CV underscore result underscores member variable.

(Refer Slide Time: 05:38)



```

cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(-mean_score, params)

```

```

0.5096674155773421 {'max_features': 2, 'n_estimators': 3}
0.38494794730392157 {'max_features': 2, 'n_estimators': 10}
0.35898284926470584 {'max_features': 2, 'n_estimators': 30}
0.4765907543572984 {'max_features': 4, 'n_estimators': 3}
0.37949047181372547 {'max_features': 4, 'n_estimators': 10}
0.3677285709422658 {'max_features': 4, 'n_estimators': 30}
0.47674223856209147 {'max_features': 6, 'n_estimators': 3}
0.3908617340686275 {'max_features': 6, 'n_estimators': 10}
0.35285364923747276 {'max_features': 6, 'n_estimators': 30}
0.47786049836601296 {'max_features': 8, 'n_estimators': 3}
0.37944690563725486 {'max_features': 8, 'n_estimators': 10}
0.35524742306644874 {'max_features': 8, 'n_estimators': 30}
0.4390253948001742 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
0.3897452818627451 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
0.4490985838779956 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
0.3858988664215686 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
0.45253914760348585 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
0.3858853860294117 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}


```

As you can notice the lowest MSE is obtained for the best parameter combination.

And you can see that these are the mean squared error and this is a combination of hyperparameters. So, in the first row, you can see that the maximum number of features to be used is 2, and the number of estimators to try is 3. In this setup, we get the mean square error of 0.5 and you can see that there are these 18 combinations that we have listed over here and for each of these combinations we have specified what is exactly the value of the hyperparameters that were tried and what is the mean squared error.

And you can see that the best mean squared error or the least mean square error is obtained for a combination of the parameter where the max feature is equal to 6 and the number of estimators equal to 30 and that value is 0.352. So, this is the best parameter setting that was obtained with the grid search.

(Refer Slide Time: 06:47)



Let's obtain the best estimator as follows:

```

1 grid_search.best_estimator_

```

```

RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                        max_depth=None, max_features=6, max_leaf_nodes=None,
                        max_samples=None, min_impurity_decrease=0.0,
                        min_impurity_split=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        n_estimators=30, n_jobs=None, oob_score=False,
                        random_state=None, verbose=0, warm_start=False)

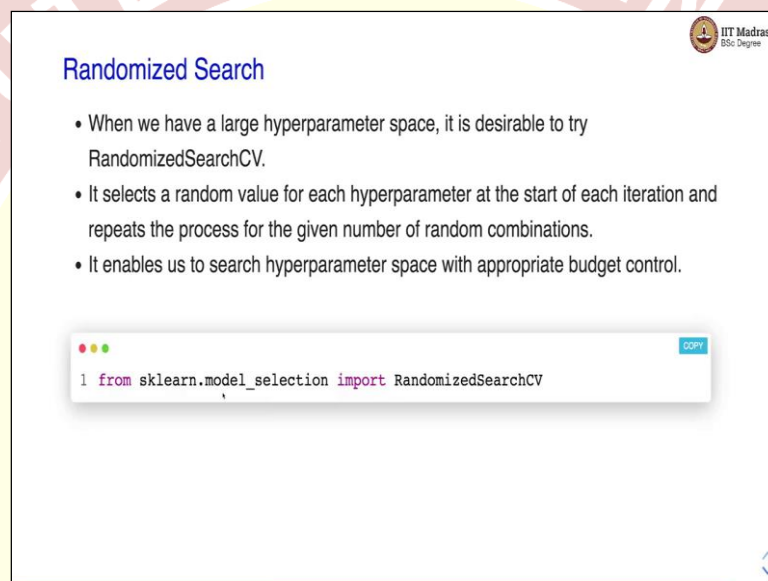
```

Note: GridSearchCV is initialized with `refit=True` option, which retrains the best estimator on the full training set. This is likely to lead us to a better model as it is trained on a larger dataset.

We can obtain the best parameter we can obtain the best estimator with the best underscore estimator underscore member variable of the grid search object. And you can see that in this best estimator maximum features are equal to set to 6 and number of estimator is set to 30 and this was the combination that was obtained with GridSearchCV. In this case we have initialized the GridSearchCV with refit equal to true option.

When we set refit equal to true the GridSearchCV retains the best estimator on the full training set this is likely to lead us to a better model as it is trained on a larger dataset.

(Refer Slide Time: 07:42)



The slide is titled "Randomized Search" and features the IIT Madras logo in the top right corner. It contains three bullet points explaining the concept of randomized search. Below the text is a code editor window showing the import statement for RandomizedSearchCV from sklearn.model_selection.

Randomized Search


- When we have a large hyperparameter space, it is desirable to try RandomizedSearchCV.
- It selects a random value for each hyperparameter at the start of each iteration and repeats the process for the given number of random combinations.
- It enables us to search hyperparameter space with appropriate budget control.

```
1 from sklearn.model_selection import RandomizedSearchCV
```

The second option that is provided by the SQL package is randomized search. When we have a large parameter space GridSearchCV can be inefficient in that case it is desirable to try RandomizedSearchCV class. RandomizedSearchCV class selects a random value for each hyperparameter at the start of each iteration and then it repeats the process for the given number of random combinations.

It enables us to search hyperparameter space with appropriate budget control. So, we can import the RandomizedSearchCV class from the model selection.

(Refer Slide Time: 08:25)



Analysis of best model and its errors

Analysis of the model provides useful insights about features. let's obtain the feature importance as learnt by the model:

```

1 feature_importances = grid_search.best_estimator_.feature_importances_

1 sorted(zip(feature_importances, feature_list), reverse=True)

```

```

[(0.2486711653610271, 'alcohol'),
 (0.14163642739406354, 'sulphates'),
 (0.12665569639367016, 'volatile acidity'),
 (0.08045272518319231, 'total sulfur dioxide'),
 (0.07275072016325315, 'density'),
 (0.05822554296729619, 'citric acid'),
 (0.05791188978825248, 'chlorides'),
 (0.057124416693656116, 'pH'),
 (0.056416454671447944, 'residual sugar'),
 (0.05388861091468478, 'fixed acidity'),
 (0.04626635046945642, 'free sulfur dioxide')]

```

- Based on this information, we may drop features that are not so important.
- It is also useful to analyze the errors in prediction and understand its causes and fix them.

The analysis of the model provides useful insights about the feature. Let us obtain the feature importance as learned by the model. So, with grid search we know that the best estimator can be obtained with best under best underscore estimator underscore member variable. We can obtain the feature importance by calling the member variable of this best estimator.

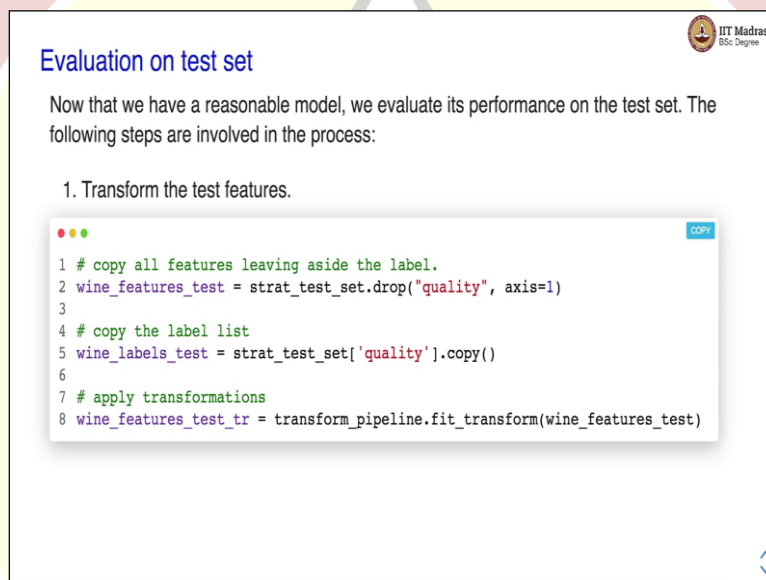
Here you can see the future importance of each of the variables and we have listed the future importance in the descending order of the importance. So, the wine quality is highly dependent on the alcohol so or the alcohol feature has is the most important feature in determining the wine quality and it has got a score of 0.24 followed by sulphate volatile acidity followed by total sulfur dioxide density, and so on.

Based on this information we may draw a few features that are not so important in the prediction of the target variable. And based on the reduced feature set we again retrain the model and follow the process we retrain the model we find the best combination of the hyperparameters using either GridSearchCV or RandomizedSearchCV depending on the size of the parameter space.

And once we obtain the best parameter we can analyze that model and find insights about the features that are important. It may also be useful to analyze the errors made by the model in prediction and understand its causes. This helps us in getting useful insights and maybe we can go back to the domain experts and consult if we can add better features to stem those errors.

So, you can see that this machine learning model development is totally an iterative process. We start with some feature set and then we train the model we perform the hyperparameter search and then analyze the model. We understand the feature importance and also look at the errors in the model. Based on the errors we come up with we go back to the experts consult them to find out how to how to take care of the error.

In order to take care of the error, we might need a new set of features or some kind of a feature transformation of the existing features we perform that and we again go back and retrain the model. And this process continues until we get the model of satisfactory quality.
(Refer Slide Time: 11:34)



Evaluation on test set

Now that we have a reasonable model, we evaluate its performance on the test set. The following steps are involved in the process:


1. Transform the test features.

```
1 # copy all features leaving aside the label.
2 wine_features_test = strat_test_set.drop("quality", axis=1)
3
4 # copy the label list
5 wine_labels_test = strat_test_set["quality"].copy()
6
7 # apply transformations
8 wine_features_test_tr = transform_pipeline.fit_transform(wine_features_test)
```

And once we are happy with the model performance once we are happy with the model that we obtained we you know we evaluate its performance on the test set. Remember we always report the performance of the model on the test set because ultimately the model has to work well it has to generalize well on the unseen data. Hence the performance of the model is reported on the test set.

While evaluating the performance of the model on the test set we need to make sure that the transformations that were applied on the training set are uniformly applied to the test set. Here we make use of the pipeline class for making sure that the transformations are applied uniformly across both the training and the test set. So we use the fit transfer method of the pipeline object on the on the test set and we obtain the transform version of the test set.

(Refer Slide Time: 12:38)




2. Use the predict method with the trained model and the test set.

```
1 quality_test_predictions = grid_search.best_estimator_.predict(  
2     wine_features_test_tr)
```

3. Compare the predicted labels with the actual ones and report the evaluation metrics.

```
1 mean_squared_error(wine_labels_test, quality_test_predictions)
```

0.3534513888888883



We use this transform version and call the predict method on the best estimator obtained to the grid search or to there or through the randomized search. And this predict method returns the predictions on the test set. Based on these predictions we compare these predictions with the actual predictions and calculate the mean squared error or any other appropriate metric based on the model at hand.

In this case, we get the mean squared error of 0.35. It is a good idea to get 95% confidence interval of the evaluation metric. It can be obtained by using stats class from scipy. We get the 95% confidence interval with stats dot interval by specifying the confidence interval and the confidence interval 95% confidence interval for mean square error, in this case, is between 0.29 and 0.41.

(Refer Slide Time: 13:50)

Step 7: Present your solution



Once we have satisfactory model based on its performance on the test set, we reach the prelaunch phase.

Before launch,

1. We need to present our solution that highlights learnings, assumptions and systems limitation.
2. Document everything, create clear visualizations and present the model.
3. In case, the model does not work better than the experts, it may still be a good idea to launch it and free up bandwidths of human experts.

Once we have a reasonable model based on its performance on the test set we reach the pre-launch phase. Before launch, we need to present our solution that highlights learning assumptions and system limitations to the group of our collaborators. As we discussed earlier machine learning model development is a collaborative activity between the model developer, domain experts, and product teams.

We need to document everything create clear visualizations and we should present our model. In case the model does not work better than the experts it may still be a good idea to launch it this will help us in freeing up precious bandwidth of the human experts.

(Refer Slide Time: 14:36)

Step 8: Launch, monitor and maintain your system



Launch

- Plug in input sources and
- Write test cases

Monitoring

- System outages
- Degradation of model performance
- Sampling predictions for human evaluation
- Regular assessment of data quality, which is critical for model performance

Maintenance

- Train model regularly every fixed interval with fresh data.

And the final step in the process is the launch step. In launch, we plug in the input sources and write test cases to make sure that the model works well end to end. Writing test cases

is quite crucial because it gives us more confidence about working different pieces together. And once the model is launched we need to monitor it closely. We need to monitor for the system outages find ways to mitigate those outages or minimize those outages.

While monitoring the model if we found that the model performance has degraded we need to have a strategy to tackle that situation. The situation can be tackled either by retraining the model and this retraining can be scheduled after some kind of a fixed time we can retrain the model let us say in every two weeks or whenever we see a drop in performance by some kind of a threshold metric then you know we can launch the retraining of the model.

So, for that, we need to first fix what is the threshold on the metric? And once that threshold is breached or once that threshold is crossed you know model performance once that threshold is crossed we need to trigger the retraining of the model. Another important thing to evaluate the model performance is to sample predictions for human evaluation. We can present predictions by the model in production to the human experts and get their feedback on how the model is performing.

We also need to carry out regular assessments of the data quality which is critical for model performance. Remember the model performance depends on the feature values and if features are not getting captured correctly it will adversely affect the model performance and the readings at an under and output that we get from the model if we consume in the downstream tasks there could be the effect that is undesirable.

Finally model maintenance we can either train the model regularly in every fixed interval with fresh data or we can also trigger the model retraining based on some kind of a threshold in the degradation of model performance. And we should also figure out how to push the trained model to production without disrupting the live path of the survey.

(Refer Slide Time: 17:35)

Summary

In this module, we studied steps involved in end to end machine learning project with an example of prediction of wine quality.

So, these were the eight steps involved in the end-to-end machine learning project. So, I hope you have better clarity about the steps that are involved in a machine learning project. I request you to follow this advice meticulously in the subsequent classes or in subsequent sessions of this particular course and also you should carry this advice into your work.

So, if you follow this advice rigorously you know you will be able to build machine learning models of high quality and you will be a successful machine learning developer. I hope you enjoyed this content and from the next class, we will start talking about know mechanics of scale and library. We start with a high-level introduction to the sklearn library and we will deep dive into different machine learning algorithms and their implementations with the sklearn library. Namaste.