


**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**

**Selection and Training of ML Models**

**(Refer Slide Time: 00:10)**



### Step 5: Select and train ML model

- It's a good practice to build a quick baseline model on the preprocessed data and get an idea about model performance.

```
1 from sklearn.linear_model import LinearRegression
2
3 lin_reg = LinearRegression()
4 lin_reg.fit(wine_features_tr, wine_labels)

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

Namaste welcome to the next video of the machine learning practice course. In this video, we will talk about how to select and train machine learning models. This is step number five in our end-to-end machine learning project. In this step, we will talk about how to select a machine learning model that can be applied to the problem at hand. So, it generally happens that when we are solving a machine learning problem and we get the data were not sure what kind of machine learning model to use.

In this case, it is a good practice to build a quick baseline model on the preprocessed data and get an idea about model performance. So, in this case in the case of wine quality prediction since the quality is a number we can use a linear regression model as a quick baseline.

**(Refer Slide Time: 01:15)**

Now that we have a working model of a regression, let's evaluate performance of the model on training as well as test sets.

- For regression models, we use mean squared error as an evaluation measure.

```
1 from sklearn.metrics import mean_squared_error
2
3 quality_predictions = lin_reg.predict(wine_features_tr)
4 mean_squared_error(wine_labels, quality_predictions)
0.4206571060060278
```

Once we train the regression model we can evaluate its performance on both the training as well as test sets. We can use mean squared error as an evaluation measure. So, in the training set, we obtain mean square error of point 0.42.

**(Refer Slide Time: 01:42)**

Let's evaluate performance on the test set.

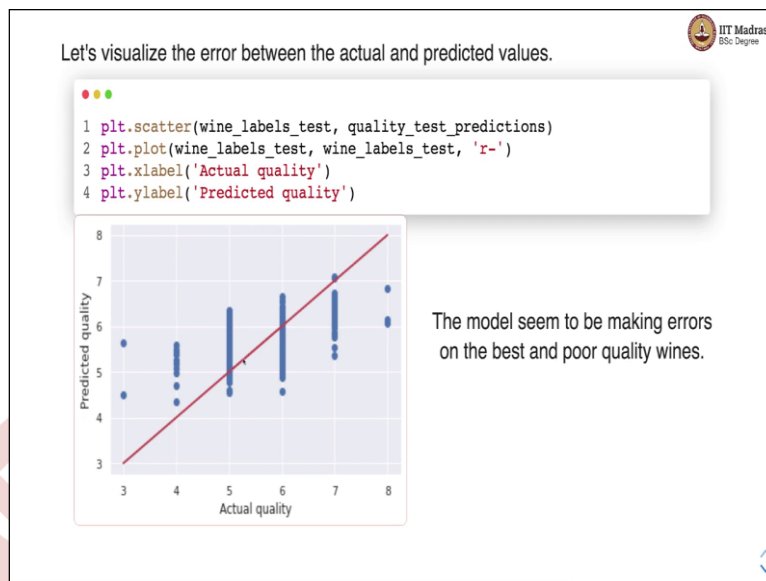
- We need to first apply transformation on the test set and then apply the model prediction function.

```
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)
9
10 # call predict function and calculate MSE.
11 quality_test_predictions = lin_reg.predict(wine_features_test_tr)
12 mean_squared_error(wine_labels_test, quality_test_predictions)
```

We evaluate the performance in the test set where we first apply the transformation on the test set same as the training set and then apply the model prediction function. So, we have the test set here we apply the same pipeline that the transformation pipeline the test set as a training set and then we calculate the mean squared error on the test set. So, these are the actual labels and these are the predicted label from our model.

We compare these two and calculate the mean square error through sklearn function. The mean squared error on the test set is 0.39.

(Refer Slide Time: 02:35)



Let us visualize the error between the actual and predicted value. So, here what we will do is we will have a scatter plot that will plot the actual wine quality which is the actual label and the predicted quality. So, for each example we can see a dot here that that shows this for this particular point the actual quality was 3 and we predicted the quality somewhere between 4 and 5.

Here the actual quality was 8 and we predicted quality which is slightly less than 7. So, this particular plot shows that the model seems to be making errors on the best and poor quality wines. So, on either extreme the model is not performing that well. On the other hand, for average quality wines model seems to be performing fairly well.

(Refer Slide Time: 03:49)

Let's try another model: DecisionTreeRegressor.

```
1 from sklearn.tree import DecisionTreeRegressor
2 tree_reg = DecisionTreeRegressor()
3 tree_reg.fit(wine_features_tr, wine_labels)
```

DecisionTreeRegressor(ccp\_alpha=0.0, criterion='mse', max\_depth=None, max\_features=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, min\_impurity\_split=None, min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, presort='deprecated', random\_state=None, splitter='best')

Notice similarity between two code snippets.

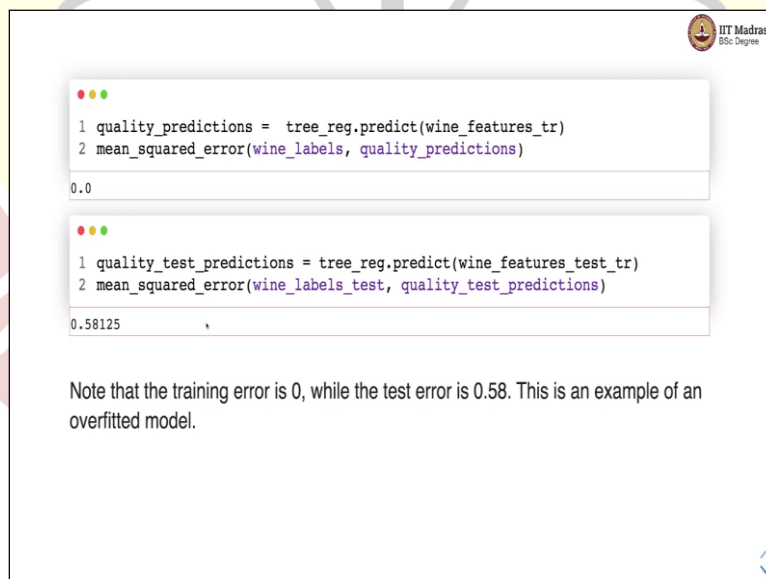
Linear regression	Decision Trees
<code>lin_reg.fit(wine_features_tr, wine_labels)</code>	<code>tree_reg.fit(wine_features_tr, wine_labels)</code>

Now that we have a quick baseline we can try to build other regression models. Decision tree regressor is another such regression model we are yet to study this in machine learning techniques course but in the later weeks of the course we are going to study decision tree regressor model. In this case what we will do is we will use the decision tree regressor as provided by the scikit-learn package.

We import the decision tree regressor and call the fit function on the training set. At this point I would like you to notice a similarity between two core snippets. In case of linear regression, we had linear regression object on which we call the fit function and in decision tree regressor we have a tree regression object on which we call the fit method. And you can see that the fit method takes the same kind of parameters these are the features of the wine and this is a label.

And this happens due to a nice code design by a scalar library both linear regressor and decision tree regressor are instances of estimator class. In the second week we will be discussing about design principles of scikit-learn library in the overview of a sklearn lecture that time some of these similarities will be more clear to you.

**(Refer Slide Time: 05:32)**



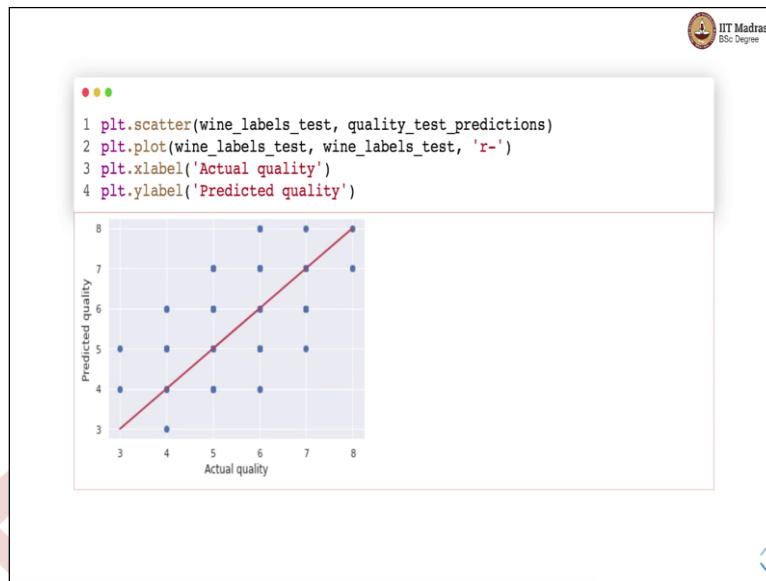
```
1 quality_predictions = tree_reg.predict(wine_features_tr)
2 mean_squared_error(wine_labels, quality_predictions)
0.0

1 quality_test_predictions = tree_reg.predict(wine_features_test_tr)
2 mean_squared_error(wine_labels_test, quality_test_predictions)
0.58125
```

Note that the training error is 0, while the test error is 0.58. This is an example of an overfitted model.

You can see that the tree regressor obtains a zero training error and the test error is 0.58. As you may notice based on our discussion that this is an example of the over fitted model.

**(Refer Slide Time: 05:54)**



Let us look at the plot of the actual quality versus the predicted quality. And we can see that the predictions are all over the place.

**(Refer Slide Time: 06:01)**

We can use cross-validation (CV) for robust evaluation of model performance.

```
1 from sklearn.model_selection import cross_val_score
```

- Cross validation provides a separate MSE for each validation set, which we can use to get a mean estimation of MSE as well as the standard deviation, which helps us to determine how precise is the estimate.
- The additional cost we pay in cross validation is additional training runs, which may be too expensive in certain cases.

```
1 def display_scores(scores):
2     print("Scores:", scores)
3     print("Mean:", scores.mean())
4     print("Standard deviation:", scores.std())
```

We can use cross validation for robust evaluation on of model performance. We have discussed cross validation in detail in machine learning techniques course. Here we can use cross validation score for evaluating the performance of the model. The cross validation provides separate a mean squared error for each validation set. We can use this separate mean squared error on different validation set to get a mean estimation of MSE as well as the standard deviation.

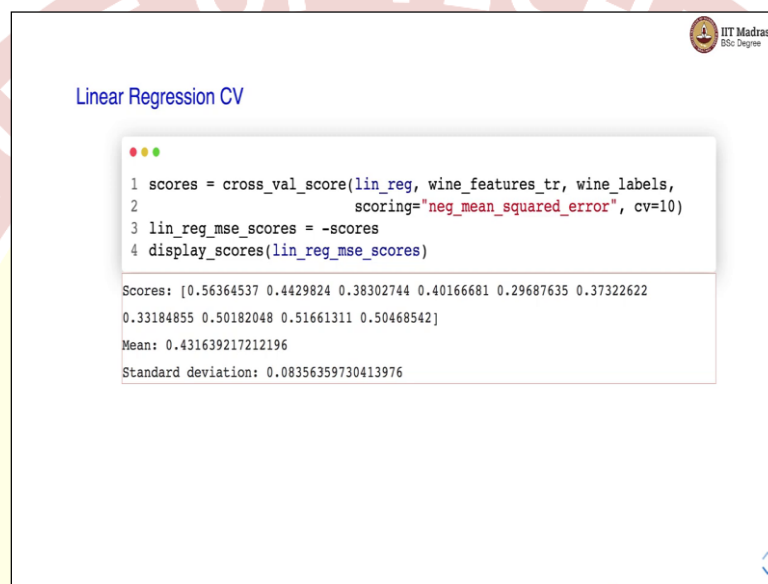
The standard deviation helps us to determine how precise is the estimate across different scores. Of course there is an additional cost that we pay with additional trading runs



because in cross validation what we do is we end up training the model on different folds. So, there are multiple training runs that are involved. This can be too expensive in certain cases.

So, this is a small code snippet where we are calculating the mean score and the standard deviation from different cross validation faults and this course actually shows the raw MSE score from different scores of cross validations.

**(Refer Slide Time: 07:31)**



```
Linear Regression CV


1 scores = cross_val_score(lin_reg, wine_features_tr, wine_labels,
2                           scoring="neg_mean_squared_error", cv=10)
3 lin_reg_mse_scores = -scores
4 display_scores(lin_reg_mse_scores)

Scores: [0.56364537 0.4429824 0.38302744 0.40166681 0.29687635 0.37322622
0.33184855 0.50182048 0.51661311 0.50468542]
Mean: 0.431639217212196
Standard deviation: 0.08356359730413976
```

Let us calculate the cross validation score on linear regressor. So, in the cross validation score function we pass the estimator object which is linear regression in this case the feature set, the label set, the scoring method and number of folds that we want to use here we use 10 folds for cross validation. We will be studying in detail what this cross validation score function does as well as what the scoring means in the subsequent weeks.

In this case we obtain different scores there are 10 different scores one MSE for each fold. So, from these 10 different values we calculate the mean which comes out to be 0.43 and the standard deviation is 0.08. You can see that the MSE varies by fold. The lowest MSE is obtained on fold number 5 right it is 0.29 and highest MSE is obtained on the first fold which is 0.56. So, the average comes out to be 0.43 and the standard deviation is 0.08.

**(Refer Slide Time: 08:55)**



### Decision tree CV

```
1 scores = cross_val_score(tree_reg, wine_features_tr, wine_labels,  
2                           scoring="neg_mean_squared_error", cv=10)  
3 tree_mse_scores = -scores  
4 display_scores(tree_mse_scores)
```

Scores: [0.6171875 0.6875 0.6328125 0.5078125 0.4609375 0.640625 0.65625 0.7109375  
0.859375 1.07874016]  
Mean: 0.6852177657480315  
Standard deviation: 0.16668343331737054

Let's compare scores of Linear regression (LinReg) and decision tree (DT) regressions:

- LinReg has better MSE and more precise estimation compared to DT.

We can apply the same kind of cross validation on the decision tree regressors. Let us look at the scores from the decision tree regressor. Again these are 10 different scores from different folds and you can see that there is a wide variability. The highest MSE is 1.07 whereas the lowest MSE seems to be 0.46 and of course it is an and this is a scoring method this is a scoring method and this scores that we have is basically negative mean squared error we take minus of that.

So, we have mean squared errors and in this case the lower mean square squared error indicates a better model. We get a mean of 0.68 and the standard deviation of 0.16. So, if we compare this course with the linear regression scores you know we find that linear regressor is has got a better MSE as well as more precise estimation compared to decision tree.

It was 0.43 in case of linear regression and the standard deviation there was 0.08 which is smaller compared to the deviation obtained from decision tree.

**(Refer Slide Time: 10:18)**



## Random forest CV

- Random forest model builds multiple decision trees on randomly selected features and then average their predictions.
- Building a model on top of other model is called *ensemble learning*, which is often used to improve performance of ML models.

```
1 from sklearn.ensemble import RandomForestRegressor
2
3 forest_reg = RandomForestRegressor()
4 forest_reg.fit(wine_features_tr, wine_labels)
5
6 scores = cross_val_score(forest_reg, wine_features_tr, wine_labels,
7                           scoring='neg_mean_squared_error', cv=10)
8 forest_mse_scores = -scores
9 display_scores(forest_mse_scores)
```

Scores: [0.36989922 0.41363672 0.29063438 0.31722344 0.21798125 0.30233828 0.27124922  
0.38747344 0.42379219 0.46229449]  
Mean: 0.34565226131889765  
Standard deviation: 0.0736322184302973

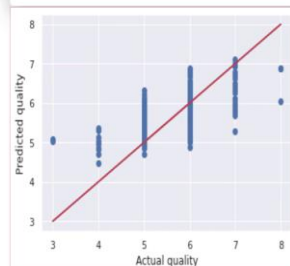
Ensemble learning methods have often found to improve the performance of machine learning models. Here we defined a random forest regressor that was that was imported from sklearn dot ensemble package. We call the fit function on the random forest aggressor in order to train the model. And then we calculate the cross validation score on the training set. Here we obtain the mean of 0.34 and standard deviation of 0.07. In case of random forest regressor you can also see a variability but that this variability is much less compared to the previous two models. Here the mean is 0.34 and the standard deviation is 0.07.

**(Refer Slide Time: 11:15)**

```
1 quality_test_predictions = forest_reg.predict(wine_features_test_tr)
2 mean_squared_error(wine_labels_test, quality_test_predictions)
```

0.34449875

```
1 plt.scatter(wine_labels_test, quality_test_predictions)
2 plt.plot(wine_labels_test, wine_labels_test, 'r-')
3 plt.xlabel('Actual quality')
4 plt.ylabel('Predicted quality')
```



Random forest looks more promising than the other two models.

- It's a good practice to build a few such models quickly without tuning their hyperparameters and shortlist a few promising models among them.

We can also calculate the mean squared error on the test sets. The mean squared error on the test set is 0.34 and just like linear regression model. So, random forest model is slightly better than the linear regression model and you know you can also see similar traits like

linear regression model. It works well on most of the wine qualities except for the extreme ones. So, the poor quality and the best quality wines are kind of not predicted that well even by the random forest model.

But random forest is more promising than linear regression and decision tree models. It is a good practice to build a few such models quickly without tuning their hyper parameters and shortlist a few promising models among them. We also need to save these models to disk in python using pickle format.

(Refer Slide Time: 12:27)

What to do next?

Model diagnosis	Remedy
Underfitting	Models with more capacity
	Less constraints/regularization
Overfitting	More data
	Simpler model
	More constraints/regularization

So, once you have the model you might be wondering what are the next steps. Once we have the model we often perform model diagnostics in order to find out whether model is suffering from underfitting or overfitting. We have discussed the ideas of underfitting and overfitting and how to overcome them in machine learning techniques course.

If your model is under fitting, we need to you know. So, basically the problem is that our model does not have enough capacity and we can fix it by using models with more capacity by using let us say something like polynomial features. If you are applying too much of regularization, then also model can under fit in that case we can reduce the amount of regularization or the constraints.

On the other hand, in case of overfitting what happens is model has excess capacity because of which model is pretty much able to memorize the training data it gets us almost perfect predictions on the training set but it fails to generalize on the test set. We can fix

the overfitting problem by getting more data or trying simple model or we can apply more constraints or regularization in order to prevent the overfitting problem.

So, once we have modeled once we have shortlisted some of the promising models the next step is fine tuning the model. We will talk about fine tuning in detail in the next video.

