MACHINE LEARNING - WORKSHEET 1

1. What is the advantage of hierarchical clustering over K-means clustering?

A) Hierarchical clustering is computationally less expensive B) In hierarchical clustering you don't need to assign number of clusters in beginning C) Both are equally proficient D) None of these

Ans: B) In hierarchical clustering you don't need to assign number of clusters in beginning.

2. Which of the following hyper parameter(s), when increased may cause random forest to over fit the data?

A) max_depth B) n_estimators C) min_samples_leaf D) min_samples_split

Ans: A) max_depth

3. Which of the following is the least preferable resampling method in handling imbalance datasets?

A) SMOTE B) RandomOverSampler C) RandomUnderSampler D) ADASYN

Ans: C) RandomUnderSampler

4. Which of the following statements is/are true about "Type-1" and "Type-2" errors? 1. Type1 is known as false positive and Type2 is known as false negative. 2. Type1 is known as false negative and Type2 is known as false positive. 3. Type1 error occurs when we reject a null hypothesis when it is actually true.

A) 1 and 2 B) 1 only C) 1 and 3 D) 2 and 3

Ans: C) 1 and 3

5. Arrange the steps of k-means algorithm in the order in which they occur: 1. Randomly selecting the cluster centroids 2. Updating the cluster centroids iteratively 3. Assigning the cluster points to their nearest centre A) 3-1-2 B) 2-1-3 C) 3-2-1 D) 1-3-2

Ans: D) 1-3-2

6. Which of the following algorithms is not advisable to use when you have limited CPU resources and time, and when the data set is relatively large? A) Decision Trees B) Support Vector Machines C) K-Nearest Neighbors D) Logistic Regression.

Ans: B) Support vector machines can take quite a bit of time to run because of their resource-intensive nature.

7. What is the main difference between CART (Classification and Regression Trees) and CHAID (Chi Square Automatic Interaction Detection) Trees? A) CART is used for classification, and CHAID is used for regression. B) CART can create multiway trees (more than two children for a node), and CHAID can only create binary trees (a maximum of two children for a node). C) CART can only create binary trees (a maximum of two children for a node), and CHAID can create multiway trees (more than two children for a node) D) None of the above

Ans: C) CART can only create binary trees (a maximum of two children for a node), and CHAID can create multiway trees (more than two children for a node)

8. In Ridge and Lasso regularization if you take a large value of regularization constant(lambda), which of the following things may occur? A) Ridge will lead to some of the coefficients to be very close to 0 B) Lasso will lead to some of the coefficients to be very close to 0 C) Ridge will cause some of the coefficients to become 0 D) Lasso will cause some of the coefficients to become 0.

Ans: C) Ridge will cause some of the coefficients to become 0

D) Lasso will cause some of the coefficients to become 0.

9. Which of the following methods can be used to treat two multi-collinear features? A) remove both features from the dataset B) remove only one of the features C) Use ridge regularization D) use Lasso regularization.

Ans: D) use Lasso regularization.

10. After using linear regression, we find that the bias is very low, while the variance is very high. What are the possible reasons for this? A) Overfitting B) Multicollinearity C) Underfitting D) Outliers

Ans: A) Overfitting

Q10 to Q15 are subjective answer type questions, Answer them briefly.

11. In which situation One-hot encoding must be avoided? Which encoding technique can be used in such a case?

Ans: Machine learning models require all input and output variables to be numeric. This means that if your data contains categorical data, you must encode it to numbers before you can fit and evaluate a model.

One hot encoding can be defined as the essential process of converting the categorical data variables to be provided to machine and deep learning algorithms which in turn improve predictions as well as classification accuracy of a model. One Hot Encoding is a common way of pre-processing categorical features for machine learning models.

Pre-processing data is an essential step before building a Deep Learning model. When creating a deep learning project, it is not always that we come across clean and well-formatted data. Therefore, while doing any operation with the data, it is mandatory to clean it and put it in a formatted way. Data pre-processing is the process of preparing the raw data and making it suitable for a machine or deep learning model and it is also the first and crucial step while creating a model. Several machine learning algorithms as well as Deep Learning Algorithms are generally unable to work with categorical data when fed directly into the model. These categories must be further converted into numbers and the same is required for both the input and output variables in the data that are categorical. We should not use the One Hot Encoding method when:

- When the categorical features present in the dataset are ordinal i.e. for the data being like Junior, Senior, Executive, Owner.
- When the number of categories in the dataset is quite large. One Hot Encoding should be avoided in this case as it can lead to high memory consumption. **In such cases Binary Encoding techniques can be used.**To fight the curse of dimensionality, binary encoding might be a good alternative to one-hot encoding because it creates fewer columns when encoding categorical variables. Then the numbers are transformed in the binary number. After that binary value is split into different columns. Binary encoding works really well when there are a high number of categories.

12. In case of data imbalance problem in classification, what techniques can be used to balance the dataset? Explain them briefly?

Ans: Imbalanced data refers to those types of datasets where the target class has an uneven distribution of observations, i.e. one class label has a very high number of observations and the other has a very low number of observations. We can better understand imbalanced dataset handling with an example. Let's assume that XYZ is a bank that issues a credit card to its customers. Now the bank is concerned that some fraudulent transactions are going on and when the bank checks their data they found that for each 2000 transaction there are only 30 Nos of fraud recorded. So, the number of fraud per 100 transactions is less than 2%, or we can say more than 98% transaction is "No Fraud" in nature. Here, the class "No Fraud" is called the **majority class,** and the much smaller in size "Fraud" class is called the **minority class**.

Resampling(Oversampling and Undersampling)

This technique is used to upsample and downsample the minority or majority class. When we are using an imbalanced dataset, we can oversample the minority class using replacement. This technique is called oversampling .

Similarly, we can randomly delete rows from the majority class to match them with the minority class which is called undersampling.

After sampling the data we can get a balanced dataset for both majority and minority classes. So when both classes have a similar number of records present in the dataset. We can assume that the classifier will give equal importance to both classes.

13. What is the difference between SMOTE and ADASYN sampling techniques?

Ans: SMOTE : What smote does is simple. First it finds the n-nearest neighbours in the minority class for each of the samples in the class. Then it draws a line between the neighbours an generates random points on the lines.

ADASYN : Its a improved version of Smote. What it does is same as SMOTE just with a minor improvement. After creating those sample, it adds a random small value to the points thus making it more realistic. In other words instead of all the sample being linearly correlated to the parent they have a little more variance in them i.e. they are bit scattered.

The key difference between ADASYN and SMOTE is that the former uses a density distribution, as a criterion to automatically decide the number of synthetic samples that must be generated for each minority sample by adaptively changing the weights of the different minority samples to compensate for the skewed distributions. The latter generates the same number of synthetic samples for each original minority sample.

14. What is the purpose of using GridSearchCV? Is it preferable to use in case of large datasets? Why or why not?

Ans: GridSearchCV is a technique for **finding the optimal parameter values from a given set of parameters in a grid**. It's essentially a cross-validation technique. The model as well as the parameters must be entered. After extracting the best parameter values, predictions are made. GridSearchCV is the process of performing hyperparameter tuning in order to determine the optimal values for a given model. GridSearchCV tries all the combinations of the values passed in the dictionary and evaluates the model for each combination using the Cross-Validation method. Hence after using this function we get accuracy/loss for every combination of hyperparameters and we can choose the one with the best

performance. For a large size dataset, **Grid Search CV time complexity increases exponentially, and hence it's not practically feasible**. One can shift to Random Search CV where the algorithm will randomly choose the combination of parameters.

15. List down some of the evaluation metric used to evaluate a regression model. Explain each of them in brief.

Ans: There are three error metrics that are commonly used for evaluating and reporting the performance of a regression model; they are:

- Mean Squared Error (MSE).
- Root Mean Squared Error (RMSE).
- Mean Absolute Error (MAE)

Mean Squared Error (MSE):

It is the average of the squared differences between the actual and the predicted values.

Lower the value, the better the regression model.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

Mean Squared Error

where,

n = total number of data points

yi = actual value

ŷi = predicted value

Its unit is the square of the variable's unit.

Here's a Scikit-learn implementation of MSE:

The example below gives a small contrived dataset of all 1.0 values and predictions that range from perfect (1.0) to wrong (0.0) by 0.1 increments. The squared error between each prediction and expected value is calculated and plotted to show the quadratic increase in squared error.

#calculate error

Err = (expected[j]-predicted[j])**2

```
from matplotlib import pyplot
from sklearn.metrics import mean_squared_error
# real value
expected = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
# predicted value
predicted = [1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0]
# calculate errors
errors = list()
for i in range(len(expected)):
# calculate error
err = (expected[i] - predicted[i])**2
# store error
errors.append(err)
# report error
print('>%.1f, %.1f = %.3f' % (expected[i], predicted[i], err))
# plot errors
pyplot.plot(errors)
pyplot.xticks(ticks=[i for i in range(len(errors))], labels=predicted)
pyplot.xlabel('Predicted Value')
pyplot.ylabel('Mean Squared Error')
pyplot.show()
```

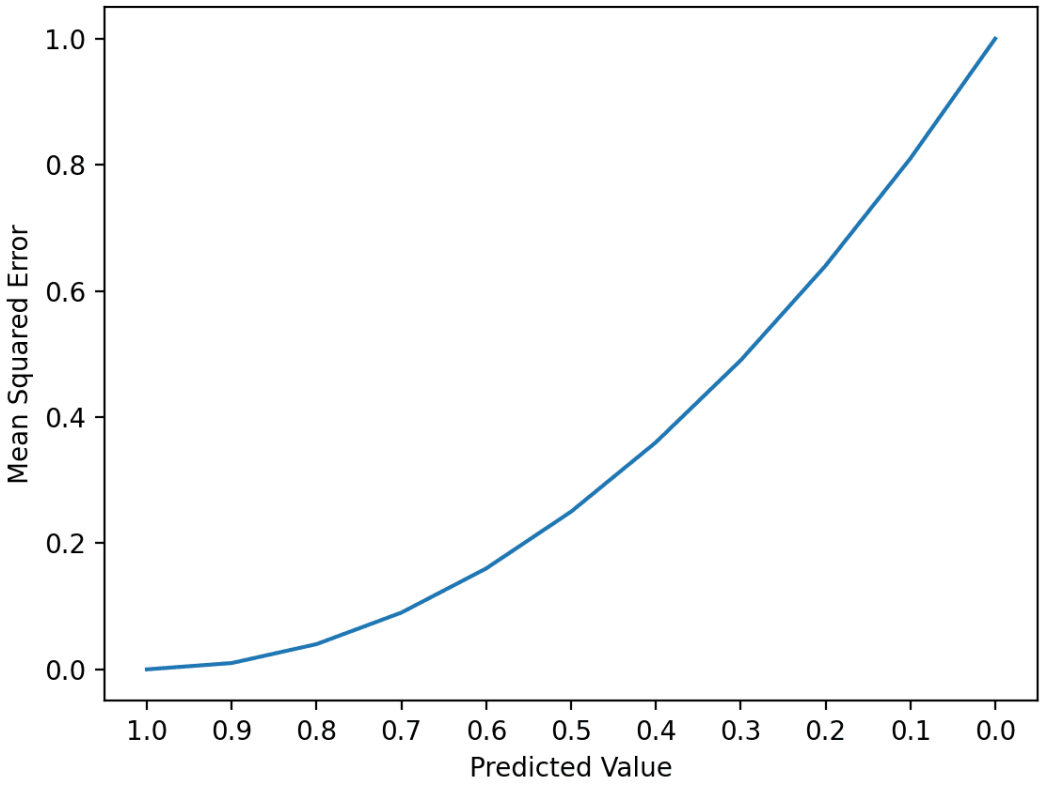
Running the example first reports the expected value, predicted value, and squared error for each case.

We can see that the error rises quickly, faster than linear (a straight line).

```
1  >1.0, 1.0 = 0.000
2  >1.0, 0.9 = 0.010
3  >1.0, 0.8 = 0.040
4  >1.0, 0.7 = 0.090
5  >1.0, 0.6 = 0.160
6  >1.0, 0.5 = 0.250
7  >1.0, 0.4 = 0.360
8  >1.0, 0.3 = 0.490
9  >1.0, 0.2 = 0.640
10 >1.0, 0.1 = 0.810
11 >1.0, 0.0 = 1.000
```

A line plot is created showing the curved or super-linear increase in the squared error value as the difference between the expected and predicted value is increased.

The curve is not a straight line as we might naively assume for an error metric.



Line Plot of the Increase Square Error With Predictions

The individual error terms are averaged so that we can report the performance of a model with regard to how much error the model makes generally when making predictions, rather than specifically for a given example.

The units of the MSE are squared units.

For example, if your target value represents "*dollars*," then the MSE will be "*squared dollars*." This can be confusing for stakeholders; therefore, when reporting results, often the root mean squared error is used instead (*discussed in the next section*).

The mean squared error between your expected and predicted values can be calculated using the mean squared error() function from the scikit-learn library.

The function takes a one-dimensional array or list of expected values and predicted values and returns the mean squared error value.

1	...

```
# calculate errors
errors = mean_squared_error(expected, predicted)
```

The example below gives an example of calculating the mean squared error between a list of contrived expected and predicted values.

```
# example of calculate the mean squared error
from sklearn.metrics import mean_squared_error
# real value
expected = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
# predicted value
predicted = [1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0]
# calculate errors
errors = mean_squared_error(expected, predicted)
# report error
print(errors)
```

Running the example calculates and prints the mean squared error.

```
0.35000000000000003
```

A perfect mean squared error value is 0.0, which means that all predictions matched the expected values exactly.

This is almost never the case, and if it happens, it suggests your predictive modeling problem is trivial.

A good MSE is relative to your specific dataset.

It is a good idea to first establish a baseline MSE for your dataset using a naive predictive model, such as predicting the mean target value from the training dataset. A model that achieves an MSE better than the MSE for the naive model has skill.

Root Mean Squared Error

The Root Mean Squared Error, or RMSE, is an extension of the mean squared error.

Importantly, the square root of the error is calculated, which means that the units of the RMSE are the same as the original units of the target value that is being predicted.

For example, if your target variable has the units "*dollars*," then the RMSE error score will also have the unit "*dollars*" and not "*squared dollars*" like the MSE.

As such, it may be common to use MSE loss to train a regression predictive model, and to use RMSE to evaluate and report its performance.

The RMSE can be calculated as follows:

- RMSE = sqrt(1 / N * sum for i to N (y_i – yhat_i)^2)

Where *y_i* is the i'th expected value in the dataset, *yhat_i* is the i'th predicted value, and *sqrt()* is the square root function.

We can restate the RMSE in terms of the MSE as:

- RMSE = sqrt(MSE)

Note that the RMSE cannot be calculated as the average of the square root of the mean squared error values. This is a common error made by beginners and is an example of Jensen's inequality.

You may recall that the square root is the inverse of the square operation. MSE uses the square operation to remove the sign of each error value and to punish large errors. The square root reverses this operation, although it ensures that the result remains positive.

The root mean squared error between your expected and predicted values can be calculated using the mean_squared_error() function from the scikit-learn library.

By default, the function calculates the MSE, but we can configure it to calculate the square root of the MSE by setting the "*squared*" argument to *False*.

The function takes a one-dimensional array or list of expected values and predicted values and returns the mean squared error value.

```
...
# calculate errors
errors =
mean_squared_error(expected,
predicted, squared=False)
```

The example below gives an example of calculating the root mean squared error between a list of contrived expected and predicted values.

```
# example of calculate the root mean
squared error
from sklearn.metrics import
mean_squared_error
# real value
expected = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0]
# predicted value
```

```
9
10
```
```
predicted = [1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0]
# calculate errors
errors = mean_squared_error(expected, predicted, squared=False)
# report error
print(errors)
```

Running the example calculates and prints the root mean squared error.

```
1
```
```
0.5916079783099616
```

A perfect RMSE value is 0.0, which means that all predictions matched the expected values exactly.

This is almost never the case, and if it happens, it suggests your predictive modeling problem is trivial.

A good RMSE is relative to your specific dataset.

It is a good idea to first establish a baseline RMSE for your dataset using a naive predictive model, such as predicting the mean target value from the training dataset. A model that achieves an RMSE better than the RMSE for the naive model has skill.

Mean Absolute Error

Mean Absolute Error, or MAE, is a popular metric because, like RMSE, the units of the error score match the units of the target value that is being predicted.

Unlike the RMSE, the changes in MAE are linear and therefore intuitive.

That is, MSE and RMSE punish larger errors more than smaller errors, inflating or magnifying the mean error score. This is due to the square of the error value. The MAE does not give more or less weight to different types of errors and instead the scores increase linearly with increases in error.

As its name suggests, the MAE score is calculated as the average of the absolute error values. Absolute or *abs()* is a mathematical function that simply makes a number positive. Therefore, the difference between an expected and predicted value may be positive or negative and is forced to be positive when calculating the MAE.

The MAE can be calculated as follows:

- MAE = 1 / N * sum for i to N abs(y_i – yhat_i)

Where *y_i* is the i'th expected value in the dataset, *yhat_i* is the i'th predicted value and *abs()* is the absolute function.

We can create a plot to get a feeling for how the change in prediction error impacts the MAE.

The example below gives a small contrived dataset of all 1.0 values and predictions that range from perfect (1.0) to wrong (0.0) by 0.1 increments. The absolute error between each prediction and expected value is calculated and plotted to show the linear increase in error.

```
...
# calculate error
err = abs((expected[i] - predicted[i]))
```

The complete example is listed below.

```
# plot of the increase of mean
absolute error with prediction error
from matplotlib import pyplot
from sklearn.metrics import
mean_squared_error
# real value
expected = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0]
# predicted value
predicted = [1.0, 0.9, 0.8, 0.7, 0.6,
0.5, 0.4, 0.3, 0.2, 0.1, 0.0]
# calculate errors
errors = list()
for i in range(len(expected)):
# calculate error
err = abs((expected[i] - predicted[i]))
# store error
errors.append(err)
# report error
print('>%.1f, %.1f = %.3f' %
(expected[i], predicted[i], err))
# plot errors
pyplot.plot(errors)
pyplot.xticks(ticks=[i for i in
range(len(errors))], labels=predicted)
pyplot.xlabel('Predicted Value')
pyplot.ylabel('Mean Absolute Error')
pyplot.show()
```

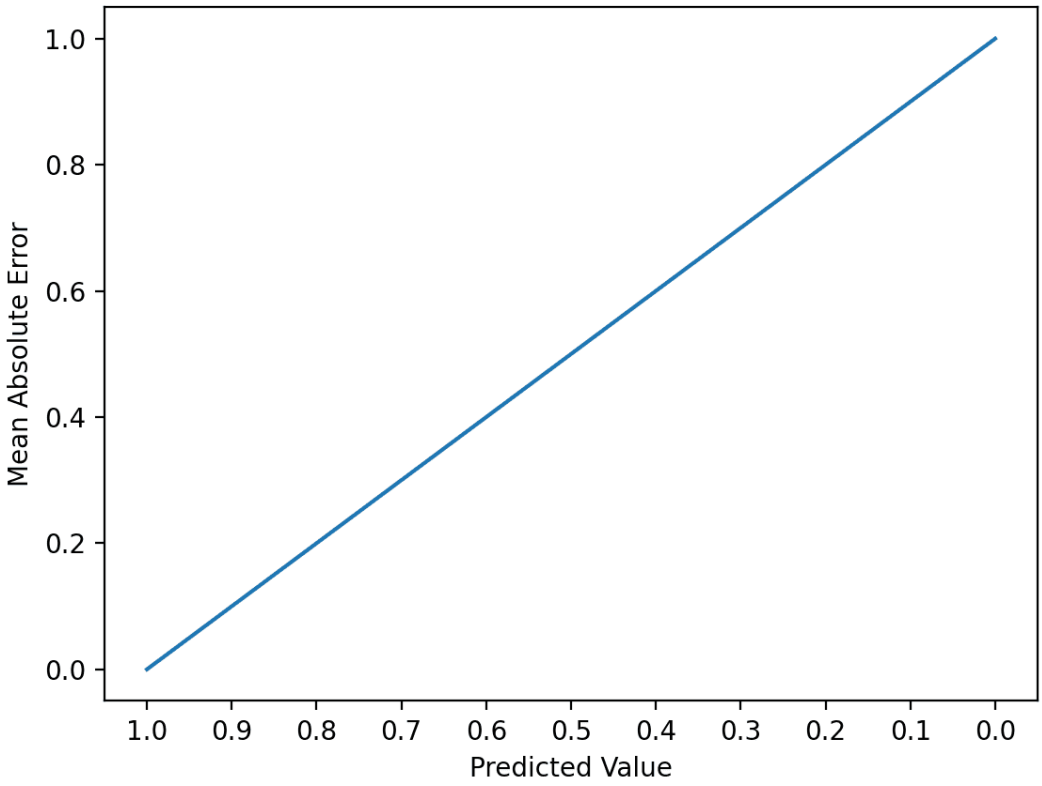
Running the example first reports the expected value, predicted value, and absolute error for each case.

We can see that the error rises linearly, which is intuitive and easy to understand.

```
>1.0, 1.0 = 0.000
>1.0, 0.9 = 0.100
>1.0, 0.8 = 0.200
>1.0, 0.7 = 0.300
>1.0, 0.6 = 0.400
>1.0, 0.5 = 0.500
```

```
7  >1.0, 0.4 = 0.600
8  >1.0, 0.3 = 0.700
9  >1.0, 0.2 = 0.800
10 >1.0, 0.1 = 0.900
11 >1.0, 0.0 = 1.000
```

A line plot is created showing the straight line or linear increase in the absolute error value as the difference between the expected and predicted value is increased.



Line Plot of the Increase Absolute Error With Predictions

The mean absolute error between your expected and predicted values can be calculated using the mean_absolute_error() function from the scikit-learn library.

The function takes a one-dimensional array or list of expected values and predicted values and returns the mean absolute error value.

```
1 ...
2 # calculate errors
3 errors =
  mean_absolute_error(expected,
  predicted)
```

The example below gives an example of calculating the mean absolute error between a list of contrived expected and predicted values.

```
# example of calculate the mean absolute error
from sklearn.metrics import mean_absolute_error
# real value
expected = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
# predicted value
predicted = [1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0]
# calculate errors
errors = mean_absolute_error(expected, predicted)
# report error
print(errors)
```

Running the example calculates and prints the mean absolute error.

```
0.5
```

A perfect mean absolute error value is 0.0, which means that all predictions matched the expected values exactly.

This is almost never the case, and if it happens, it suggests your predictive modeling problem is trivial.

A good MAE is relative to your specific dataset.It is a good idea to first establish a baseline MAE for your dataset using a naive predictive model, such as predicting the mean target value from the training dataset. A model that achieves a MAE better than the MAE for the naive model has skill.