Introduction to Machine Learning

Hot topic right?

Everyone wants to learn about `machine learning` and `AI` these days and I am no exception. But what is `machine learning`? What is `AI`? Let's find out together. I'm `Md. Rishat Talukder` and let's get into it.

- LinkedIn
- YouTube
- gtihub
- Gmail
- discord

In this article I'll be going over the basic concepts of `machine learning` and steps to build a `machine learning` model.

What is Machine Learning?

Machine learning is the process of teaching a computer to learn from data without being explicitly programmed to do so.

One thing that is emphasized here is that the computer is learning from the `data` itself and not from program made by a human.

So, data is the key of machine learning.

We give computer some data(a huge amount of data) and it should be able to learn the patterns from it and make `predictions` that are close to if not exactly the same as the human made predictions.

It is a mathematical process of simulating how a human brain works and how a computer works.

Machine learning can be categorized into two types:

- **Supervised learning**
- **Unsupervised learning**

Supervised Learning

Traditinal ML algorithms are `supervised learning` algorithms.

Supervised learning is the process where you have `labeled data` and the task for the computer is to learn the patterns from the data and make predictions.

Every dataset should have a `target` or `output` column and the data inside that should be `labeled`. Like, a person's age, height, weight, etc.

Unsupervised Learning

This is where the term `AI` comes into play.

A machine learning model that does not have `labeled data` and figures out the patterns from the data itself is called `unsupervised learning`.

`Agents` like `gpt, gemini, claude` etc. learns patterns from the data itself and makes predictions. Even though I like to call these `psuedo AI` they are very close to the real `AI` and are just a simulation of a `AI` trained on a `large dataset`.

Machine Learning Lifecycle

Machine learning is a `cycle` of learning and then using the model to make predictions.

This process can be broken down into the following steps:

1. **Data Collection**: Gather a large amount of data to train the model on.
2. **Data Preprocessing/Cleaning**: Clean and preprocess the data to make it suitable for training.
3. **Splitting the Data**: Split the data into training and testing sets.
4. **Training the Model**: Train the machine learning model on the training data.
5. **Evaluating the Model**: Evaluate the model's performance on the testing data.
6. **Making Predictions**: Use the trained model to make predictions on new data.

These are the steps every machine learning model goes through. It is a cycle of learning and then using the model to make predictions.

Now, let's talk more in depth about the steps of the machine learning lifecycle.

Data Collection

In this step, we gather a large amount of data to train the model on. The data can be in the form of text, images, audio, video, etc.

In this step we also have to figure out what we are trying to predict.

So, we need to collect data that has a `target` or `output` that should be labeled as well.

Data Preprocessing/Cleaning

Next step is to `clean` and `preprocess` the data to make it suitable for training.

This where our data analysis and visualization comes into play.

We use our knowledge to make the data cleaner and more suitable for training.

Also this is the most lengthy step of the machine learning lifecycle. Because if you don't analyze the data well, you might end up with a dataset that has a imbalanced distribution of the data and can cause `overfitting` or `underfitting` or `bias` in the model.

> I'll talk about overfitting and underfitting as we progress through this article series.

Splitting the Data

Next step is to `split` the data into training and testing sets.

So, if we use all our data into training a ML model, what do we use to evaluate the model?

We need some input data so that we can test the model on data that the model has never seen before.

This way we can have a proper evaluation of the model.

So, we must split the data into `training` and `testing` sets. Sometimes, we might split the data into `training`, `validation` and `testing` sets. Te validation set comes in hands when we want to check the performance of the model on data that the model has never seen before.

> In neural networks we use `validation` set to check the learning progress and `testing` set to check the final performance of the model.

Training the Model

Next step is to `train` the machine learning model on the training data.

This is where the magic happens.

We choose a model that is suitable for our problem and train it on the training data.

There are a lot of models available in `scikit-learn` and `tensorflow` that we can choose from and also there are multiple algorithms to solve the same problem but can have different performance on different datasets.

Evaluating the Model

After the model is trained, we need to `evaluate` the model's performance on the testing data.

There can be two types of evaluation:

- **Accuracy**

- **Loss**

I'll talk more about this later on this article.

Making Predictions

Now that we have a trained model, we can use it to make predictions on new data. If the performance of the model is a satisfactory, we can use it to make predictions on new data and use the predictions to make decisions or take actions.

Evaluating a Model

Evaluating a model is `comparing` the prediction of the model with the actual output.

Let's say you have a dataset like below:

Input	Output
1	x
2	y
3	x
4	x
5	x
6	y
7	x
8	y
9	y
10	x

> You can see that the data is labels with `x` and `y`. Only two classes.

Now, let's say we have a model that predicts the output based on the input. Let's say the model predicts the output like below:

Input	Prediction
1	x
2	y
3	x
4	y (wrong)
5	x
6	x (wrong)
7	x
8	y
9	y

Input	Prediction
10	x

Now, If we compare the prediction of the model with the actual output, we can get the accuracy of the model.

Actual Output	Prediction
x	x
y	y
x	x
x	y
x	x
x	x
x	x
y	y
y	y
x	x

We can clearly see that out of 10 predictions, 8 are correct and 2 are incorrect.

So, the accuracy of the model is $\frac{number\ of\ correct\ predictions}{total\ number\ of\ predictions}$.

So, the accuracy of the model is $\frac{8}{10} = 0.8$.

So, the accuracy of the model is 80%.

Which is good. But this doesn't give us the whole picture.

Let's do a thought experiment. Let's say you have a data set with 100 samples(rows) and in the input you have two classes. 1. x and 2. y.

> If you have only two classes in a data set, it is refered to as `binary classification`.

Now, In the target class let's say you have `99 x` and only `1 y`.

Now, let's say you have a model that is trained on this data and predicts the output. And it gives you a output of `100 x`.

So, you have only one wrong prediction. So, the accuracy should be $\frac{99}{100} = 0.99$.

So, the accuracy of the model is 99%.

Which is amazing right?

Nope, it is not. What do you think will happend you test the model with unseen data that was not part of the training data and has only `y` class. Do you think the model can predict the `y` class correctly?

Of course not, This is a classic example of `overfitting` or `bias` in the model.

I like to remember this with a analogy.

You learned to ride a bicycle so well you cannot ride a motorcycle.

This can be caused by different reasons. One of them in imbalanced data like the thought experiment above.

So, the point is that even though you models accuracy is 99%, it is not a good model and will not perform well on new data.

That's why we need other way to measure the performance of a model and that's when metrics come into play.

- `Accuracy.`
- `Precision.`
- `Recall.`
- `F1 score.`

These are the metrics that will help us understand our model performance better.

So, let's talk about them one by one.

Confusion Matrix

`Confusion matrix` is a table that shows the number of `true positives`, `true negatives`, `false positives` and `false negatives` of a classification.

and helps us decribe the overall performance of a model.

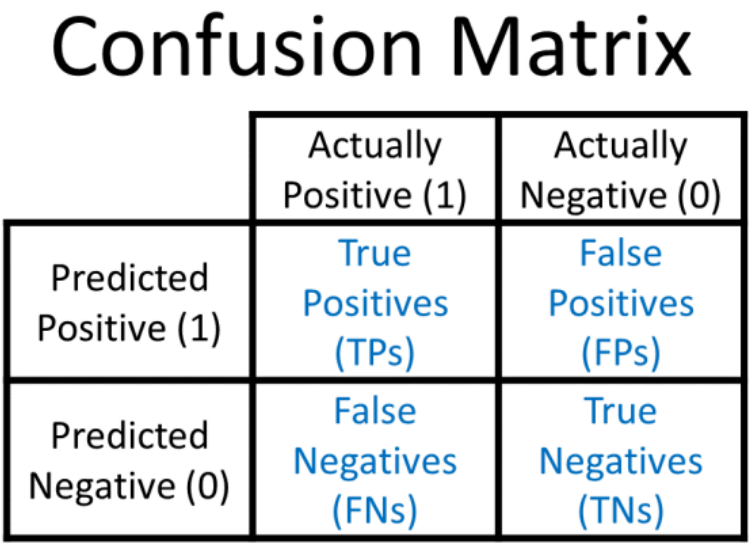
Let's see with an example.

Actual Output	Prediction
True	True
True	False (wrong)
False	True (wrong)
False	False
True	True
True	False (wrong)
False	True (wrong)
False	False
True	True
False	False
True	True
False	False
True	True

Actual Output	Prediction
False	False
True	True

> We can clearly see that out of 15 predictions, 11 are correct and 4 are incorrect.

Let's make a confusion matrix.



A confusion matrix looks like this. In the Left you have predicted output and on the top you have actual output.

Now, we have `two classes` : True and False

So, we need to make two columns for the classes and two rows for the classes of actual and predicted output respectively.

So, let's make a confusion matrix.

	Actual Positive(True)	**Actual Negative(False)**
Predicted Positive(True)	True Positive (TP)	False Positive (FP)
Predicted Negative(False)	False Negative (FN)	True Negative (TN)

Now, let's fill the cells with the values. But what are they?

- `True Positive (TP)` : The number of samples that are actually `True` and were correctly predicted as `True` .
- `False Positive (FP)` : The number of samples that are actually `False` but were incorrectly predicted as `True` .
- `False Negative (FN)` : The number of samples that are actually `True` but were incorrectly predicted as `False` .
- `True Negative (TN)` : The number of samples that are actually `False` and were correctly predicted as `False` .

> Now you understnad why it is called a `confusion matrix`. Yeah, that's why.

So, we just have to count.

Let count the values. From the table above. Let's count how many `values was True and was predicted True`. It's `6`. So, `True positive` count is `6`.

Now, let's count how many `values was True and was predicted False`. Count is `2`. So, `False Negative` count is `2`.

How many `values was False and was predicted True` ? -> 2, So, `False Positive` count is `2`.

How many `values was False and was predicted False` ? -> 5, So, `True Negative` count is `5`.

So, if we look at the confusion matrix. It looks like this.

	Actual Positive(True)	**Actual Negative(False)**	**Total**
Predicted Positive(True)	True Positive = 6	False Positive (FP) = 2	8
Predicted Negative(False)	False Negative (FN) = 2	True Negative (TN) = 5	7
Total	8	7	15

> The total will always be equal to the number of samples in the dataset.

This is the complete confusion matrix.

Now, we need to analyze the confusion matrix.

What is the `True positive and negetive` count represent?

> These two values represent the number of times the model correctly predicted the positive and negative class. So, for positive values it is right `6` times and for negative values it is right `5` times. So, in total it is `11` times out of `15` predictions. This means the accuracy of the model is `11/15 = 0.73`. So, the accuracy of the model is 73%.

We can also re-write the accuracy formula.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

So, the accuracy of the model is `0.73`.

This is how we calculate the accuracy of a model from a confusion matrix.

Precision

Remember the thought experiment?

Precision describes ratio of true positives to `all positives`.

> As we have only two classes(true/false) precision referes to the ratio of true positives to all positives.

> But if we had other classes(x/y) we can calculate the precision for each class.

Precision represents the correctness of predicted class.

$$Precision = \frac{TP}{TP + FP}$$

so, $\frac{6}{6+2} = 0.75$

So, the precision of the model is `75%`.

Recall

Recall is almost same but it is describes ability to detect positives.

$$Recall = \frac{TP}{TP + FN}$$

> For multiple classes this describes the ability to detect each class. The higher the recall, the better the model can detect a class.

This also helps us understand bias.

F1 score

Finally, the formula that combines precision and recall.

$$F1\ score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

`F1 score` is a `weighted average` of precision and recall.

> The hermonic mean of precision and recall is called `F1 score`. It describes the balance between precision and recall.

So, if the model is good at detecting a class and also has a high precision, then the `F1 score` will be high. Other wise, the `F1 score` will be low if any of the metrics is low.

So, it describes the overall performance of the model.

So, from this discussion we can say that,

A model can have a high accuracy, but if it is not good at detecting a class or has a low `f1 score`, then it is not a good model.

That's how you actually understand the confusion matrix.

Now, let's talk about how we can evaluate the performance of a regression model.

Regression Metrics

`Regression metrics` are used to evaluate the performance of a regression model.

A model that predicts a continuous value rather than a class is called a regression model.

Let's say you have a comparison between the actual and predicted values of a regression model.

Actual	Predicted
1	1.01
2	1.95
3	2.99
4	4.65
5	5.12

In the actual values, the values are continuous and in the predicted values, the values are also continuous.

And we can clearly see that the predicted values are slightly higher or lower than the actual values.

But how do we know if the model is good or not?

Let's `add up` all the differences between the actual and predicted values. Will that be a good way to understand the model performance?

The number might be big or small.

> If the sum of the differences is big, then the model is not good. If the sum of the differences is small, then the model is good.

We CAN say that, right? But how do we know if the number we are seeing is big or small in the context of the actual and predicted values?

So, this is when we use `regression metrics` to evaluate the performance of a regression model.

Let's go through them one by one.

Mean Absolute Error (MAE)

We calculate the sum of the absolute difference between the actual and predicted values and then divide it by the number of samples.

This way the number we are seeing is big or small in the context of the actual and predicted values.

`Mean Absolute Error` is the average of the absolute difference between the actual and predicted values.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

Here, `n` is the number of samples, and y_i is the actual value and $\hat{y}_i$ is the predicted value.

We can think of this as the average distance between the actual and predicted values.

For exmaple, let's say you are trying to predict `candy price` of a grocery store and you have a `mean absolute error` of `10` dollers which is not good because candy price is actually cheap but your model predicted it to be expensive.

But let's say you are trying to predict `house price` of a city and you have a `mean absolute error` of `1000` dollers which is good because house price is actually expensive but your model predicted it to be close to the actual price.

But this `metric` has an issue.

Thid doesn't punish the `larger` errors more than the `smaller` errors.

That's why we use `mean squared error` instead.

Mean Squared Error (MSE)

We calculate the sum of the `squared` difference between the actual and predicted values and then divide it by the number of samples.

Exactly same to `MAE` but this time we square the difference between the actual and predicted values.

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

So, we are getting the average of the `squared` difference between the actual and predicted values.

Why this is good?

It's because now if there is a large error from the model it will be `punished` more than if there is a small error and the average of the `squared` difference between the actual and predicted values will be larger.

And as always BUUUUUUUT!

As we are squaring the difference between the actual and predicted values if the values represent a measuresment like meters or kilograms or dollers then the measurement will also be squared.

So, the `mean squared error` can look like this, $MSE = 39.65\ dollers^2$.

This can cause confusion.

So, what we can do is to take the square root of the `mean squared error` which will cancel out the squaring.

Root Mean Squared Error (RMSE)

This is the square root of the `mean squared error`.

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

That's it!

The big difference are punished and we have a value that can be used to easily understand the performance of the model.

That's how we evaluate the performance of a regression model.

These matrics are also refered to as `loss functions`.

And that's it for this discussion.

Final Words

Tata!\n",