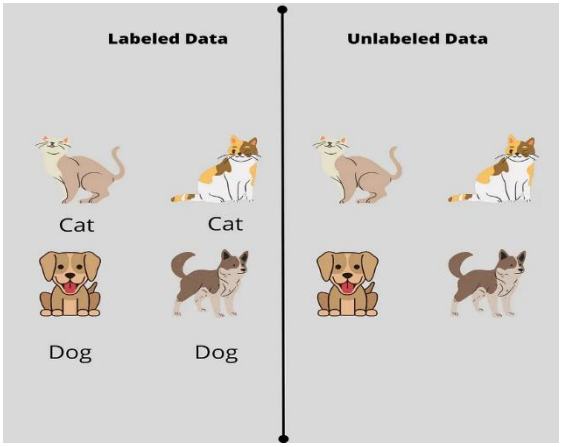
Machine Learning Lecture 2 (KNN)

By Mohamed Hafez

Supervised Learning

- Supervised learning is a type of machine learning where an algorithm learns to make predictions or decisions based on labelled input data. In this approach, the algorithm is trained using a dataset that contains both input examples and their corresponding correct output or labels.



- Once the algorithm is trained on this labelled dataset, it can use the learned patterns and relationships to make predictions or classify new, unseen data. For example, if the algorithm is trained on a dataset of emails labelled as spam or not spam, it can then accurately classify new emails as spam or not based on the patterns it has learned.

- Supervised learning algorithms include popular techniques like linear regression, decision trees, support vector machines, and neural networks. These algorithms are widely used in various domains such as image recognition, natural language processing, and recommendation systems.

Classification loss function & Mean Squared Error

- A classification loss function is a mathematical measure used to evaluate how well a classification model performs. It quantifies the difference between predicted class labels and the actual class labels in a dataset. The goal is to minimize this difference, indicating the model's ability to correctly classify instances.

- Mean Squared Error (MSE): Mean Squared Error is a common regression loss function used to measure the average squared difference between predicted and actual values. It assesses how well a regression model approximates the true relationship between features and targets.

- For example, imagine you have a dataset of houses with their corresponding values. You can train a regression model to predict the price of a house based on its features like square footage, number of bedrooms, etc. The model's predicted prices can be compared to the actual prices using MSE.

- MSE is calculated by taking the average of the squared differences between predicted and true values. The formula is as follows:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

- Here, "y" denotes the true value and "ŷ" represents the predicted value. The difference between the true and predicted values is squared, and the average is taken over all examples. The goal is to minimize the MSE, indicating a better fit of the model to the data.

Squared Loss, Linear Loss, Asymmetric Loss

- Squared Loss: Squared loss, also known as mean squared error (MSE), is a type of loss function that measures the average squared difference between the predicted and actual values in a regression problem. It penalizes larger differences more than smaller ones.

- For example, let's say we have a regression model that predicts housing prices. If the predicted price for a particular house is $300,000, but the actual price is $350,000, the

squared loss would be calculated as (350,000 - 300,000) ^2 = 2,500,000. The squared loss is higher when there is a larger difference between the predicted and actual values.

- Linear Loss: Linear loss, also known as absolute loss or L1 loss, is another type of loss function used in regression. It measures the average absolute difference between the predicted and actual values. Unlike squared loss, linear loss penalizes differences linearly without squaring them.
- For example, if the predicted price for a house is $300,000 and the actual price is $350,000, the linear loss would be calculated as |350,000 - 300,000| = 50,000. Linear loss gives equal weight to all errors, regardless of their magnitude.

- Asymmetric Loss: An asymmetric loss function assigns different penalties to the errors depending on whether the prediction is too high or too low. It takes into account the specific context of the problem where overestimating or underestimating has different consequences.

- For example, let's consider a scenario where a model predicts whether a patient has a rare disease. Misclassifying a positive case as negative could have more severe consequences than misclassifying a negative case as positive. In this case, an asymmetric loss function can assign a higher penalty to false negatives to minimize the risk.

Precision/Recall/F1

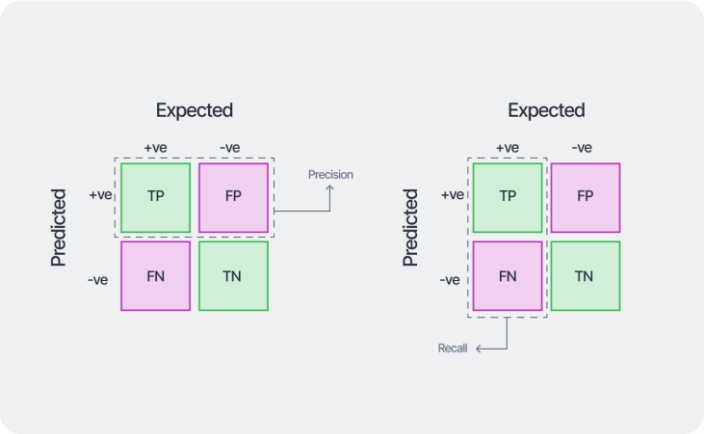
- Precision, recall, and F1 are metrics commonly used to evaluate the performance of a classification model.

- Precision: Precision measures the proportion of true positive predictions among all positive predictions made by the model. It focuses on the precision of the model in predicting positive instances correctly.

- For example, let's say we have a model that predicts whether an email is spam or not. If the model predicts that 100 emails are spam, and 90 of them are actually spam, then the precision would be 90%. It indicates how precise the model is in identifying spam emails correctly.

- Recall: Recall measures the proportion of true positive predictions among all actual positive instances. It focuses on the model's ability to identify positive instances correctly.

- In the email spam example, if there were actually 120 spam emails, but the model predicted only 90 of them as spam, then the recall would be 75%. It indicates how well the model captures all the positive instances.



- F1 Score: The F1 score is a combination of precision and recall into a single metric. It aims to provide an overall assessment of a model's performance by considering both precision and recall.

- The F1 score is calculated using the harmonic mean of precision and recall, which gives equal importance to both metrics. It ranges between 0 and 1, where a value of 1 represents perfect precision and recall.

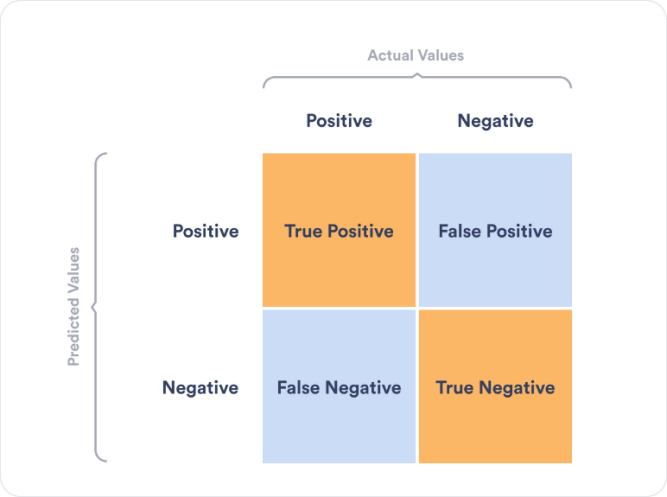
- For instance, if a model has a precision of 80% and a recall of 75%, then the F1 score would be calculated as follows: F1 Score = 2 * (Precision * Recall) / (Precision + Recall) F1 Score = 2 * (0.80 * 0.75) / (0.80 + 0.75) F1 Score = 0.774

- The F1 score helps to strike a balance between precision and recall, indicating how well the model performs overall in its ability to identify positive instances correctly while minimizing false positives.

Confusion Matrix

- The confusion matrix is a table that helps evaluate the performance of a classification model by summarizing the model's predictions against the actual values. It helps us understand the number of correct and incorrect predictions made by the model.

- Imagine you have a model that predicts whether an email is spam or not. The confusion matrix would look like this:



- Here's what the terms in the confusion matrix mean:

- True Positive (TP): The model correctly predicts an email as spam when it is actually spam.
- False Positive (FP): The model incorrectly predicts an email as spam when it is not spam.
- False Negative (FN): The model incorrectly predicts an email as not spam when it is actually spam.
- True Negative (TN): The model correctly predicts an email as not spam when it is not spam.

- The confusion matrix helps us quantify the model's performance by showing the number of correct and incorrect predictions for each class. It allows us to calculate metrics like accuracy, precision, recall, and F1 score, which provide deeper insights into the model's effectiveness.

Binary Classification & Multi-class classification

- Binary classification is a type of classification problem where the task is to classify data into one of two categories or classes. For example, predicting whether an email is spam or not spam is a binary classification problem. The goal is to assign each instance to one of the two classes based on certain features or characteristics.

- On the other hand, multi-class classification is a classification problem where the task is to classify data into more than two categories or classes. For instance, classifying emails into categories like spam, promotions, or personal would be a multi-class classification problem. The goal is to assign each instance to one of the multiple classes based on the given features or attributes.

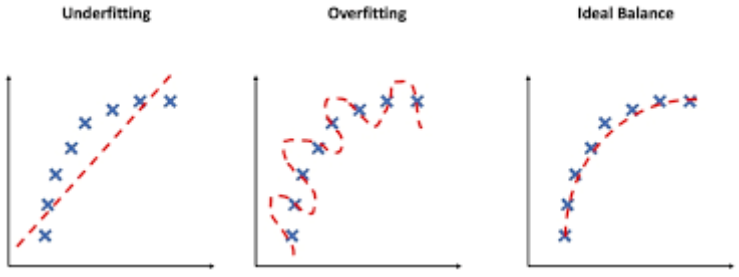
- In simple terms, binary classification involves dividing data into two categories, while multi-class classification deals with dividing data into more than two categories.

Overfitting & underfitting

- In simple terms, overfitting and underfitting are two common problems in machine learning models.

- Overfitting occurs when a model learns the training data too well and starts to perform poorly on new, unseen data. It happens when the model captures noise or irrelevant patterns in the training data, making it less effective in making accurate predictions. For example, imagine a model that tries to predict if a student will pass a test based on the number of hours they studied. If the model overfits, it may learn the exact study hours for each student in the training data, including any random fluctuations or outliers that are not actually indicative of passing the test. As a result, when presented with new students, the overfit model may make incorrect predictions because it relies too heavily on those specific patterns from the training data.

- On the other hand, underfitting occurs when a model fails to capture the underlying patterns in the training data and performs poorly even on the training data itself. It happens when the model is too simple or lacks complexity to represent the true patterns in the data. For example, let's say we have a model that tries to predict housing prices based on the size of the house. If the model underfits, it may assume that all houses have an average price regardless of their size. This simplistic model would fail to capture the nuanced relationship between house size and price, resulting in poor predictions.



- Both overfitting and underfitting are undesirable because they impact the model's ability to generalize well to new data. The goal is to find the right balance where the model is neither too complex nor too simplistic, resulting in accurate predictions on both the training data and unseen data.

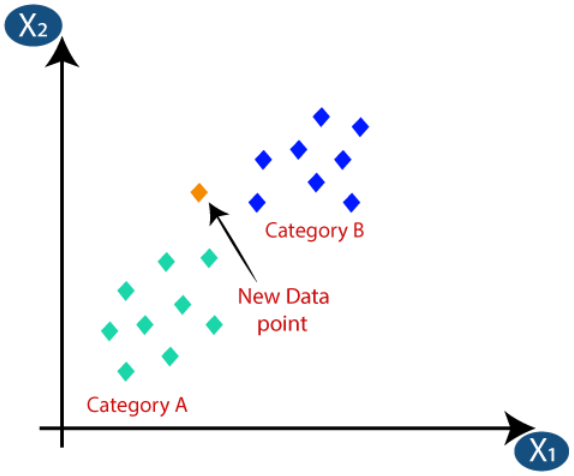
KNN Algorithm

- The K-Nearest Neighbors (KNN) algorithm is a simple and intuitive classification algorithm that makes predictions based on the similarity of data points.

- In simple terms, KNN works like this: Imagine you have a dataset of people classified as "tall" or "short" based on their height and weight. When a new person comes along, you want to predict if they are tall or short.

- With KNN, you first choose a value for K, which represents the number of nearest neighbors to consider. Then, the algorithm looks at the K nearest neighbors of the new person based on their height and weight. For example, if K=3, the algorithm will consider the three people in the dataset who are most similar in terms of height and weight to the new person.

- Next, it looks at the labels of those nearest neighbors. Let's say that two of the three nearest neighbors are classified as "tall", and one is classified as "short". In this case, KNN takes the majority vote and classifies the new person as "tall" because the majority of the nearest neighbors are tall.



- In essence, the KNN algorithm uses the similarity of known data points to make predictions for new, unseen data points. It assigns labels based on the majority class of the nearest neighbors. KNN is a simple yet effective algorithm and can be applied to various classification tasks.

Euclidean Distance

- The Euclidean distance is a mathematical concept that measures the straight-line distance between two points in space. In the context of the K-Nearest Neighbors (KNN) algorithm, the Euclidean distance is used to determine the similarity between data points.

- In simple terms, let's say we have a dataset of houses with their sizes (in square feet) and prices (in dollars). We want to use the KNN algorithm to predict the price of a new house based on its size.

- To do this, the algorithm calculates the Euclidean distance between the size of the new house and the sizes of existing houses in the dataset. For example, let's say the new house is 1800 square feet, and we have two existing houses in the dataset with sizes of 1500 square feet and 2000 square feet.

- The Euclidean distance is calculated as the square root of the sum of the squared differences of the corresponding dimensions. In this case, the distance between the new house and the first existing house is: sqrt ((1800 - 1500)^2) = 300

- And the distance between the new house and the second existing house is: sqrt ((1800 - 2000)^2) = 200

- The KNN algorithm then considers the K nearest neighbors with the smallest Euclidean distances. For example, if K=3, it would consider the three closest houses to the new house based on their sizes.

- Once the nearest neighbors have been identified, the algorithm looks at their corresponding prices to make a prediction. For example, if two of the three nearest neighbors have prices of $300,000 and $400,000, and one has a price of $500,000, the KNN algorithm would take the majority vote and predict the price of the new house to be $300,000.

Difference between the KNN Algo and KNN Classifier

- Yes, there is a difference between the KNN algorithm and the KNN classifier, although they are related.

- The KNN (K-Nearest Neighbors) algorithm is a machine learning algorithm that can be used for both classification and regression tasks. It is used to predict the value of a new data point by looking at the K nearest data points to that point. The KNN algorithm is a type of instance-based method, where the model is trained on specific instances or data points instead of generalized patterns.
- On the other hand, the KNN classifier is a specific implementation of the KNN algorithm for classification tasks. In this case, the KNN algorithm is used to classify new data points based

on the majority class of their K nearest neighbors. The KNN classifier is one of the simplest and most popular classification algorithms in machine learning.

- In summary, the KNN algorithm is a more general machine learning algorithm that can be used for both classification and regression, while the KNN classifier is a specific implementation of the KNN algorithm for classification tasks.

KNN Classifier

- The KNN (K-Nearest Neighbors) classifier is a simple yet powerful classification algorithm that makes predictions based on the similarity between data points. In other words, it classifies new data by comparing it to known data points and assigning it to the majority class of its nearest neighbors.

- To explain it simply, let's say we have a dataset of fruits categorized as "apple" or "orange" based on their weight and color. We want to use the KNN classifier to predict the category of a new fruit based on its weight and color.

- Here's how the KNN classifier works:

- Choose a value for K, which represents the number of nearest neighbors to consider.
- Calculate the Euclidean distance between the new fruit's weight and color and the weight and color of each fruit in the dataset.
- Select the K fruits with the shortest Euclidean distances.
- Look at the categories (apple or orange) of those K nearest neighbors.
- Take the majority vote of the categories to classify the new fruit. For example, if 3 out of the K nearest neighbors are apples and 2 are oranges, the KNN classifier would classify the new fruit as an apple.

- In summary, the KNN classifier makes predictions based on the majority class of the K nearest neighbors. It is a flexible and intuitive algorithm that can be applied to various classification tasks, allowing us to classify new data based on its similarity to existing labeled data points.

KNN Hyperparameter

- Hyperparameters are adjustable settings that we specify during the training of a machine learning model. For example, the KNN algorithm has a hyperparameter called K, which determines the number of nearest neighbors used in the classification of new data points.

- In simple terms, hyperparameters are like the settings on a camera that you adjust before taking a picture. Just as different camera settings can produce different photos, different hyperparameters can lead to different performance of a machine learning model.

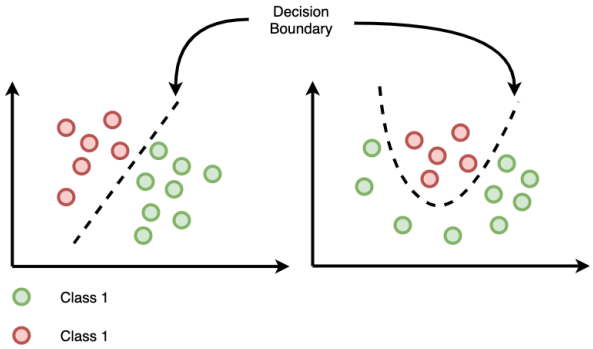
- Let's take the example of the KNN algorithm and its hyperparameter K. If we set K=1, the algorithm will classify new data based on the closest neighbor. This may result in an overfit model, where the algorithm is too focused on individual data points and not generalizing well to new data.

- On the other hand, if we set K=10, the algorithm will consider a larger number of neighbors when making predictions. This may result in an underfit model, where the algorithm is too generalized and not considering the details of individual data points in the decision process.

- Therefore, finding the optimal value for K is important in the performance of the KNN algorithm, and this requires tuning the hyperparameter. This can be done by assessing the algorithm's performance on validation data for different values of K and selecting the optimal value that results in the best performance on new, unseen data.

Decision Boundaries

- Decision boundaries are imaginary lines or surfaces that separate different classes or categories in a machine learning model. They represent the dividing line between different predictions made by the model.



- To explain it simply, let's say we have a binary classification problem where we want to predict whether an email is spam or not based on the length of the email content and the number of exclamation marks it contains.

- In this case, the decision boundary is the line or boundary that separates the emails predicted as spam from those predicted as not spam. It's like drawing a line on a graph to separate two different groups of points.

- Here's an example: Let's assume that emails with a length greater than 200 words and more than 5 exclamation marks are predicted as spam, while emails below these thresholds are predicted as not spam.

- In this scenario, the decision boundary is a line on a two-dimensional graph, with length on one axis and the number of exclamation marks on the other axis. The boundary separates the spam emails (above and to the right of the line) from the non-spam emails (below and to the left of the line).

- The decision boundary can be more complex and non-linear if we have more features or use more sophisticated machine learning algorithms. It could be a curved line, a plane, or even a higher-dimensional surface in more complex problems.

- In summary, decision boundaries are the lines or surfaces that separate different categories or classes in a machine learning model. They help determine which category or class a new data point belongs to based on its position relative to the boundary.

Decision Boundaries of KNN

- The decision boundary of a KNN (K-Nearest Neighbors) classifier refers to the line or region that separates different classes in the feature space. It represents the dividing line that the KNN algorithm uses to classify new data points based on their proximity to the class labels of the training data.

- In the context of a KNN classifier, the decision boundary determines how the algorithm assigns class labels to new data points by considering the majority class among its K nearest neighbors. For example, if the decision boundary is a straight line in a two-dimensional feature space, then the KNN classifier will assign data points on one side of the line to one class and data points on the other side to another class.

- Essentially, the decision boundary of a KNN classifier is a specific instance of the more general concept of decision boundaries in classifiers. It represents the region or line that separates classes based on the patterns and relationships in the training data.

- So, while the concept of the decision boundary remains the same across classifiers, the decision boundary of a KNN classifier specifically refers to the line or region that separates classes based on the K nearest neighbors.

Variation on KNN

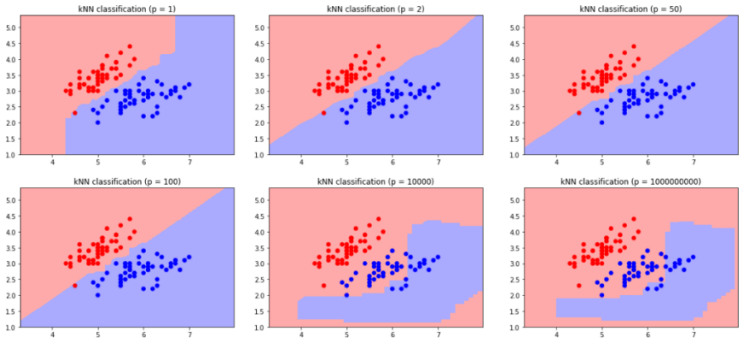
- The variation in KNN (K-Nearest Neighbors) refers to the changes in the behavior or performance of the KNN algorithm when different values are chosen for the "K" parameter.

- To explain it simply, let's consider a classification problem where we want to predict whether an animal is a cat or a dog based on its weight and height. The KNN algorithm uses the K nearest neighbors to classify the animal.

- The variation in KNN occurs when we try different values for K. For example, if we set K to 3, the algorithm will consider the three nearest neighbors to a given animal to make the classification. If two out of the three nearest neighbors are cats, and one is a dog, then the algorithm will predict that the animal is a cat.

- However, if we change the K value to 5, the algorithm will now consider the five nearest neighbors. If three of the five nearest neighbors are dogs and two are cats, then the algorithm will predict that the animal is a dog.

- The choice of the K value can significantly impact the performance of the KNN algorithm. A smaller K value can make the model more sensitive to local variations in the data, potentially leading to overfitting. On the other hand, a larger K value can make the model more robust to noise but may also blur the decision boundaries.



- The variation in KNN arises from the different choices of K and the influence it has on the classification decisions. By experimenting with different values of K, we can observe how the algorithm responds and select the appropriate value that provides the best performance for the specific problem at hand.

Issues on KNN

While the K-Nearest Neighbors (KNN) algorithm is simple and intuitive, it does have some potential issues and limitations. Here are a few common issues with KNN:

- Sensitivity to irrelevant features: KNN considers all features equally when determining nearest neighbors. If some features are not relevant to the classification task or contain noise, they can dominate the distance computation and lead to less accurate results. For example, if we include a completely random feature that has no correlation with the class labels, it may introduce unnecessary noise and affect the KNN's performance.
- Impact of the K value: The choice of the K value in KNN is critical and can significantly influence the algorithm's behavior. A smaller K value can make the model sensitive to outliers or data with high variance, leading to overfitting. On the other hand, a larger K value can smooth out the decision boundaries and potentially result in underfitting by oversimplifying the model.
- Computationally expensive: During the prediction phase, KNN calculates the distances between the new data point and all training data. This process can be computationally expensive, especially with large datasets or high-dimensional feature spaces, as it requires calculating distances for each training instance.
- Imbalanced class distribution: KNN assumes that all classes have equal representation in the training data. If there is a significant imbalance in the class distribution, with one class having far more instances than the others, it can lead to biased predictions and poor performance for the minority classes.
- Curse of dimensionality: KNN can suffer from the "curse of dimensionality" when working with high-dimensional feature spaces. As the number of dimensions increases, the available data becomes sparse, and the notion of "nearest neighbors" becomes less meaningful. This can result in degraded performance and increased computational complexity.
- To overcome these issues, it is essential to preprocess the data, select the relevant features, normalize the feature values, and choose an appropriate K value using techniques like cross-validation. Additionally, considering other machine learning algorithms that may better handle specific challenges can be a good alternative to KNN.