UNIT 2

21. What are the measures to compute the skewness of the distribution?

Skewness is a statistic that measures the asymmetry of a distribution. Given a sequence of values, xi,

- The **sample skewness** is:

$$g_1 = m_3/m_2^{3/2}$$

$$m_2 = \frac{1}{n}\sum_i (x_i - \mu)^2$$

$$m_3 = \frac{1}{n}\sum_i (x_i - \mu)^3$$

Where m_2 is the mean squared deviation (variance), and m_3 is the mean cubed deviation.

Negative skewness indicates that a distribution "skews left;" that is, it extends farther to the left than the right. Positive skewness indicates that a distribution skews right. If there are any outliers, they have a disproportionate effect on g_1.

- **Pearson's median skewness coefficient** is an alternative measure of skewness and is given by:

$$g_p = 3(\mu - \mu_{1/2})/\sigma$$

Where μ is the mean and $\mu_{1/2}$ is the median of the distribution.

Extreme values have more effect on the mean than the median, so in a distribution that skews left, the mean is less than the median. This statistic is robust, which means that it is less vulnerable to the effect of outliers.

22. Define a class that represents random variables with an exponential distribution.

A random variable represents a process that generates a random number. We can define a class to represent a random variable with an any distribution by providing a method called *'generate'* that uses the given distribution (random process eg: exponential/normal, etc) to generate a random number.

The following class generates random numbers from an exponential distribution.

```
import random

class RandomVariable(object):
    """Parent class for all random variables."""
    pass

class Exponential(RandomVariable):
  def __init__(self, lam):
    self.lam = lam

  def generate(self):
    return random.expovariate(self.lam)

a = Exponential(0.5)
a.generate()
```

Example Output: 2.819811073280088

The constructor takes a parameter *lam*, that represents λ in the following equation for the PDF of an exponential distribution:

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases}$$

23. State the PDF of exponential distribution and explain how CDF can be used to calculate it.

The PDF of exponential distribution is:

$$f(x,\lambda) = \lambda e^{-\lambda x}$$

The derivative of a CDF is called a probability density function (PDF). Thus given the CDF, by computing it's derivative, we can obtain the PDF.
Thus for exponential distribution, the CDF is given by:

$$F(x,\lambda) = 1 - \lambda e^{-\lambda x}$$

When we compute the derivative of the CDF, we get the PDF of the distribution. The PDF is actually the probability density, ie. probability per unit of x, so in order to get the probability mass, we need to integrate the PDF over values of X. ie. to calculate the probability of X (the random variable) taking a value between *a* and *b* , we need to evaluate the integral

$$P(a \leq X \leq b) = \int_a^b PDF_X(x)dx$$

The CDF is the integral of the PDF, and hence we can calculate the same using the CDF as:

$$P(a \leq X \leq b) = CDF_X(a) - CDF_X(b)$$

24. Derive the convolution of two exponential distributions.

Let X and Y be two random variables with exponential distributions.

The convolution of the PDFs of X, Y is given by

$$PDF_Z = PDF_X * PDF_Y$$

$$PDF_Z(z) = \int_{\infty}^{-\infty} PDF_X(x)PDF_Y(z-x)dx = \int_{\infty}^{-\infty} \lambda e^{-\lambda x}\lambda e^{-\lambda(z-x)}dx$$

Since PDF of Exponential distribution for all x < 0 is 0, we can adjust the limits of the above integral as

$$PDF_Z(z) = \lambda^2 e^{-\lambda z} \int_z^0 e^{-\lambda x + \lambda x}dx$$

$$PDF_Z(z) = \lambda^2 e^{-\lambda z} \int_z^0 1dx$$

$$PDF_Z(z) = \lambda^2 z e^{-\lambda z}$$

Thus we get the convolution of PDFs two exponential distributions as the PDF of an Erlang Distribution with parameter K = 2. The equation below is the PDF of an Erlang Distribution

$$f(x; k, \lambda) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!} \quad \text{for } x, \lambda \geq 0,$$

25. Discuss the importance of Central Limit Theorem.

Normal distributions are closed under linear transformations, hence, If we add values drawn from two normal distributions, the resulting values will also be normally distributed. However this property does not necessarily hold for other distributions. The Central Limit Theorem, states that if we add up a large number of values drawn from almost any distribution, the resulting values converge to a normal distribution.

Specifically if the mean and variance of the distribution of values is μ and $\boldsymbol{\sigma}$, and the values are drawn independently from a distribution with a finite mean and variance, then the distribution of the sum of values is approximately $N(n\mu, n\boldsymbol{\sigma})$.

The Central Limit Theorem explains, at least in part, the prevalence of normal distributions in the natural world. Most characteristics of animals and other life forms are affected by a large number of genetic and environmental factors whose effect is additive. The characteristics we measure are the sum of a large number of small effects, so their distribution tends to be normal. The theorem is a key concept in probability theory because it implies that probabilistic and statistical methods that work for normal distributions can be applicable to many problems involving other types of distributions. Even when we don't know the original distribution from which the data was drawn, assuming it is a normal distribution will still help find useful insights from the data.

26. Explain Hypothesis testing and the meaning of Type I and Type II errors.

Hypothesis testing is the process of determining whether an observed effect is statistically significant or not. The underlying logic is similar to a proof by contradiction. To prove a mathematical statement, A, we assume temporarily that A is false. If that assumption leads to a contradiction, we conclude that A must actually be true. Similarly, we say that an effect is not significant. This is called the **null hypothesis**, and based on that assumption we calculate the probability of the apparent effect, known as the **p-value**, if the p-value is low enough, we conclude that the null hypothesis is unlikely to be true.

In hypothesis testing, there are two kinds of errors that occur, Type I errors and Type II errors.

- **Type I** errors are also called **false positives**, which is when we conclude that a hypothesis is true (accept), while in reality it is false. In other words we consider an effect to be significant, when it was actually due to chance.

- **Type II** errors are also called **false negative**, which is when we conclude that a hypothesis is false (reject), while in reality it is true. In other words we consider an effect to be due to chance, when it was actually significant.

27. What are the different ways to measure the results of Hypothesis testing.

In hypothesis testing, we start with a null hypothesis, H_0 that is the Hypothesis that the effect is not real. We calculate the **p-value**, which is $P(E \mid H_0)$ where E is an effect as big or bigger than the apparent effect. We then compare the **p-value** with a threshold **α**

There are several ways to interpret the result of a hypothesis test:

- Classical: In classical hypothesis testing, if a **p-value** is less than α, then the effect is statistically significant, but we can't conclude that it's real. This formulation is careful to avoid leaping to conclusions, but it is not very useful.

- Practical: In practice, people are not so formal. In most science journals, researchers report **p-values** and readers interpret them as evidence that the apparent effect is real. The lower the p-value, the higher their confidence in this conclusion.

- Bayesian: This approach tries to find $P(H_A| E)$, where H_A is the hypothesis that the effect is real. By Bayes's theorem:

$$P(H_A \mid E) = P(H_A)\frac{P(E \mid H_A)}{P(E)}$$

  where $P(H_A)$ is the prior probability of HA before we saw the effect, $P(E \mid H_A)$ is the probability of seeing E, assuming that the effect is real, and P(E) is the probability of seeing E under any hypothesis, Since the effect is either real or it's not,

$$P(E) = P(E \mid H_A) P(H_A) + P(E \mid H_0) P(H_0)$$

28. Walk through the Chi Square Test with an example.

Let us take an example of rolling a 6 sided Die, 60 times. If a table of outcomes with frequencies in this experiment is given, and we need to check if the die is biased or not, we can use the Chi-Square test to test the significance of getting the given frequencies, and if the conclusion is that the effect (seeing the given frequencies) is statistically significant, then the we can conclude that the Hypothesis is true, and the Die is biased.

Say the outcome frequencies are:

Outcome	1	2	3	4	5	6
Frequency	8	9	19	6	8	10

The sum of frequencies given add up to 60.

We make a null hypothesis that the die is fair.

We calculate the Expected Frequency from a theoretical standpoint = 1/6 * 60 = 10

If E is the expected frequency of an outcome and O is the observed frequency, we compute the Chi-Square statistic, by calculating

$$\frac{(E-O)^2}{E}$$

for each observation, and sum the result to get

Outcome	1	2	3	4	5	6
Frequency	8	9	19	6	8	10
$(E-O)^2/2$	0.4	0.1	8.1	1.6	0.4	0.0

$$\sum \frac{(E-O)^2}{E} = 10.6$$

Which is approximately Chi-Square distributed with k-1 degrees of freedom (k is the number of categories/outcomes). Thus we get the chi-square test statistic for this problem as as $\chi^2 =$ 10.6 which has 5 degrees of freedom. Reading the value from a Chi-Square table, for 10.6, at 5 degrees of freedom, we get the P-value to be 0.05 Thus the null hypothesis that the dice is fair is false, and we conclude that the die is biased.

29. How are MSE and MLE different in helping find the best estimator?

When finding the best estimator, we need to decide on what the goal is. If we are trying to minimize errors, we can go with an estimator that minimizes Mean Squared Error (MSE). However if the goal is to maximize the chances of getting the right value from the estimator, we need to go with an estimator that has the highest chances/likelihood of being right. This is called the Maximum Likelihood Estimator (MLE).

If there are no outliers, the sample mean minimizes the mean squared error (MSE). Thus when we want to estimate the value of the mean of the sample, if we use the sample mean as an estimator, and there are no outliers, it minimizes:

$$MSE = \frac{1}{m}\sum(\bar{x} - \mu)^2$$

Where m is the number of times we sample the distribution (not n the sample size used to compute sample mean).

30. Discuss the biased and unbiased estimators for a distribution.

Unbiased estimators are those which have a mean error of 0, over multiple samples drawn from the distribution. While a biased estimator is one for which the error between the estimate and the actual value is non-zero over multiple samples.

Thus we can say that the bias of an estimator is the difference between this estimator's expected value and the true value of the parameter being estimated. Bias can also be measured with respect to the median, rather than the mean (expected value), in which case one distinguishes median-unbiased from the usual mean-unbiasedness property.

The sample mean is a typical example of an unbiased estimator. However the sample variance in its naive form is a biased estimator of variance, which can be corrected by a scale factor;

Suppose X1, ..., Xn are independent and identically distributed (i.i.d.) random variables with expectation μ and variance σ^2. If the sample mean and uncorrected sample variance are defined as

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \qquad S^2 = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2$$

then then S^2 is a biased estimator of σ^2. We can obtain an unbiased estimator by dividing by a (n-1) instead of n,

$$S^2_{n-1} = \frac{1}{n-1}\sum(x_i - \bar{x})^2$$

This is the more generally used definition for S^2 as it is an unbiased estimator of the population variance.

31. Confidence intervals are better than point estimates. Justify.

A major advantage of using confidence intervals is that we get a range of values with a known probability of capturing the population parameter. Thus a 95% confidence interval can help us make a claim with 95% confidence that the interval will include the true population parameter.

Point estimates are generally based on the sample, and give a single value for the population parameter without claims on probability of being accurate.We also have no information of the upper and lower bounds of the population parameter, which is given in a confidence interval.

Another aspect that makes confidence intervals better is that they provide a range of values that the population parameter can take along with how likely each value in the range is. Confidence intervals can also be easily computed by simulations, even though they can be hard to calculate analytically.

32. How can Bayesian estimation be used to calculate confidence intervals?

Let us say we have a sample from a random distribution, X. If we assume that the population parameter, eg. λ for exponential distribution, was drawn from a uniform distribution. Then for some interval (a, b) which is the lower and upper bound for λ, we divide the interval into equal sized bins, and define a hypothesis H_i for each bin, that is the hypothesis that the actual value of λ falls into the i^{th} bin..

Then for each hypothesis we calculate the likelihood of getting the sample X given the hypothesis H_i, as

$$P(X \mid H_i) = \prod expo(\lambda_i, x)$$

where expo(λ, x) is the PDF of exponential distribution with parameter λ evaluated at x

Then by Bayes theorem, the posterior distribution is given by

$$P\ (H_i \mid X) = P(X \mid H_i)\ P(H_i)\ /f$$

where f is the normalizing factor (sum of all $P(X \mid H_i)\ P(H_i)$)

Given a posterior distribution, it is easy to compute a confidence interval. For example, to compute a 90% CI, we can use the 5th and 95th percentiles of the posterior.

33. Define the statistical significance of correlation and two ways of calculating it.

Correlation quantifies both the strength and direction of the linear relationship between two measurement variables. The value of computing correlation is that we get a single number that measures the strength of linear relationships between two variables. It indicates how close the values fall to a straight line. The range of values of correlation is between -1 to 1.

A statistically significant relationship is one that is large enough to be unlikely to have occurred in the sample if there's no relationship in the population. The issue of whether a result is unlikely to happen by chance is an important one in establishing cause-and-effect relationships from experimental data.

A challenge in measuring correlation is that the variables we want to compare might not be expressed in the same units or they may come from different distributions.

Thus two common ways to compute correlation are:

- Transform all values to standard scores. This leads to the Pearson coefficient of correlation
- Transform all values to their percentile ranks. This leads to the Spearman coefficient of correlation

If X is a series of values, x_i, we can convert to standard scores by subtracting the mean and dividing by the standard deviation.

34. Why is covariance not used alone? How can it be used to define correlation?

Covariance is a measure of the tendency of two variables to vary together. If we have two series, X and Y, their deviations from the mean are

$$dxi = xi - \mu X$$
$$dyi = yi - \mu Y$$

where μX is the mean of X and μY is the mean of Y. If X and Y vary together, their deviations tend to have the same sign.
Covariance is the mean of these products:

$$Cov(X,Y) = 1/n \sum dx_i \, dy_i$$

where n is the length of the two series (they have to be the same length). Covariance is useful in some computations, but it is seldom reported as a summary statistic because it is hard to interpret. Among other problems, its units are the product of the units of X and Y. So the covariance of weight and height might be in units of kilogram-meters, which doesn't mean much.

Correlation is a description of how two variables are related. For example if we are comparing heights of students, we can also infer from the same that students who are taller will tend to be heavier as well.

35. Compare Pearson's correlation with Spearman's rank correlation.

Covariance is useful in some computations, but it is seldom reported as a summary statistic because it is hard to interpret. Among other problems, its units are the product of the units of X and Y. One solution to this problem is to divide the deviations by σ, which yields standard scores, and compute the product of standard scores:

$$p_i = \frac{x_i - \mu_x}{\sigma_x} \frac{y_i - \mu_y}{\sigma_y}$$

And then compute the mean of these products, to get

$$\rho = \frac{1}{n} \sum p_i$$

And then we can factor it out to write

$$\rho = \frac{Cov(X, Y)}{\sigma_x \sigma_y}$$

Which is the pearson's correlation coefficient. Pearson's correlation works well if the relationship between variables is linear and if the variables are roughly normal. But it is not robust in the presence of outliers.

Spearman's rank correlation is an alternative that mitigates the effect of outliers and skewed distributions. To compute Spearman's correlation, we have to compute the rank of each value, which is its index in the sorted sample. For example, in the sample {7, 1, 2, 5} the rank of the value 5 is 3, because it appears third if we sort the elements. Then we compute Pearson's correlation for the ranks.

36. Explain how Least Squares Fit can be used to calculate the slope of a relationship.

Correlation coefficients measure the strength and sign of a relationship, but not the slope. There are several ways to estimate the slope; the most common is a linear least squares fit. A "linear fit" is a line intended to model the relationship between variables. A "least squares" fit is one that minimizes the mean squared error (MSE) between the line and the data .

Suppose we have a sequence of points, Y, that we want to express as a function of another sequence X. If there is a linear relationship between X and Y with intercept α and slope β, we expect each y_i to be roughly $\alpha + \beta * x_i$.

Here's how least squares fit can be used :
1. Compute the sample means, $\bar{x}$ and $\bar{y}$, the variance of X, and the covariance of X and Y.
2. The estimated slope is

$$= \frac{\text{Cov (X, Y)}}{\text{Var (X)}}$$

3. And the intercept is

$$\hat{\alpha} = \bar{y} - \hat{\beta}\ \bar{x}$$

37. Demonstrate the interpretation of a sample R-squared value to determine goodness of fit.

One way to evaluate a model, to determine how good it is, is to measure its predictive power. In the context of prediction, the quantity we are trying to guess is called a dependent variable and the quantity we are using to make the guess is called an explanatory or independent variable. To measure the predictive power of a model, we can compute the coefficient of determination, more commonly known as "R-squared":

$$R^2 = 1 - \frac{Var(\varepsilon)}{Var(Y)}$$

To understand what R2 means, suppose (again) that you are trying to guess someone's weight. If you didn't know anything about them, your best strategy would be to guess y' ; in that case the MSE of your guesses would be

$$MSE = \frac{1}{n}\sum(\bar{y} - y_i)^2 = Var(Y)$$

But if I told you their height, you would guess ˆa + ˆb xi; in that case your MSE would be Var(e).

$$MSE = \frac{1}{n}\sum(\hat{\alpha} + \hat{\beta}x_i - y_i)^2 = Var(\varepsilon)$$

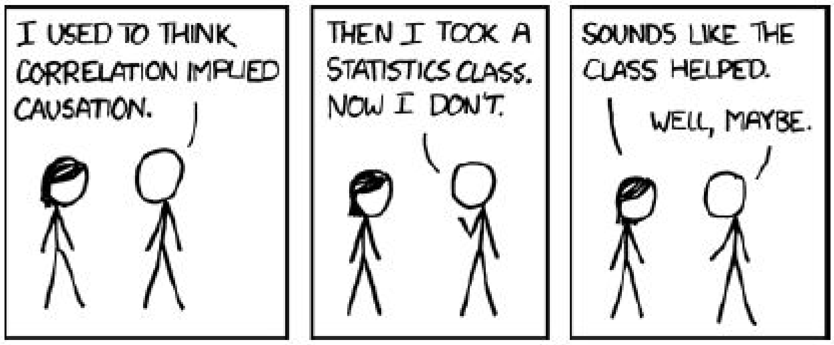
So the term Var(e)/Var(Y) is the ratio of mean squared error with and without the explanatory variable, which is the fraction of variability left unexplained by the model. The complement, R2, is the fraction of variability explained by the model.

If a model yields R2 = 0.64, you could say that the model explains 64% of the variability, or it might be more precise to say that it reduces the MSE of your predictions by 64%.

38. Discuss the complex relationship between correlation and causation.

In general, a relationship between two variables does not tell you whether one causes the other, or the other way around, or both, or whether they might both be caused by something else altogether. This rule can be summarized with the phrase "Correlation does not imply Causation,"

Example


I USED TO THINK CORRELATION IMPLIED CAUSATION.
THEN I TOOK A STATISTICS CLASS. NOW I DON'T.
SOUNDS LIKE THE CLASS HELPED.
WELL, MAYBE.

39. Why is randomised controlled trial used to determine a causal relationship?
So ways to provide evidence of causation?:
1. Use time. If A comes before B, then A can cause B but not the other way around. The order of events can help us infer the direction of causation, but it does not preclude the possibility that something else causes both A and B.
2. Use randomness. If you divide a large population into two groups at random and compute the means of almost any variable, you expect the difference to be small. This is a consequence of the Central Limit Theorem (so it is subject to the same requirements). If the groups are nearly identical in all variable but one, you can eliminate spurious relationships.
This works even if you don't know what the relevant variables are, but it works even better if you do, because you can check that the groups are identical.

These ideas are the motivation for the randomized controlled trial, in which subjects are assigned randomly to two (or more) groups: a treatment group that receives some kind of intervention, like a new medicine, and a control group that receives no intervention, or another treatment whose effects are known. A randomized controlled trial is the most reliable way to demonstrate a causal relationship, and the foundation of science-based medicine Unfortunately, controlled trials are only possible in the laboratory sciences, medicine, and a few other disciplines. In the social sciences, controlled experiments are rare, usually because they are impossible or unethical.

40. How can linear regression be used to infer causal relationships?

In some cases it is possible to infer causal relationships using regression analysis. A linear least squares fit is a simple form of regression that explains a dependent variable using one explanatory variable. There are similar techniques that work with arbitrary numbers of independent variables.
Example: The NSFG(National Survey of Family Growth) says that first babies tend to be lighter than others. But birth weight is also correlated with the mother's age, and mothers of first babies tend to be younger than mothers of other babies. So it may be that first babies are lighter because their mothers are younger.
To control for the effect of age, we could divide the mothers into age groups and compare birth weights for first babies and others in each age group. If the difference between first babies and others is the same in each age group as it was in the pooled data, we conclude that the difference is not related to age. If there is no difference, we conclude that the effect is entirely due to age. Or, if the difference is smaller, we can quantify how much of the effect is due to age.

UNIT 3

41. Compare supervised and unsupervised learning approaches in machine learning.

In Supervised learning, data is labeled and the machine learning algorithms learn to predict the output from the input data. In other words, supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output - Y = f(X). The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data. It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance. In supervised learning, creating a dataset of inputs and outputs is often a laborious manual process, but the algorithms are well understood and their performance is easy to measure.

In unsupervised learning, data is unlabeled and the algorithms learn the inherent structure from the input data. In other words, unsupervised learning is where you only have input data (X) and no corresponding output variables. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. These are called unsupervised learning because unlike supervised learning above there is no correct answers and there is no teacher. Algorithms are left to their own devices to discover and present the interesting structure in the data. While unsupervised learning does not require the laborious task of labelling data, they are usually harder to understand and evaluate.

42. Distinguish between classification and regression.
There are two major types of supervised machine learning problems i.e classification
and regression.

In classification, given a set of input variables, the goal is to predict a class label, which is a choice from a predefined list of possibilities. Classification is sometimes separated into binary classification, which is the special case of distinguishing between exactly two classes, and multiclass classification, which is classification between more than two classes. Thus classification deals with outputs that are discrete categorical values
Eg. Classifying emails as either spam or not spam - binary classification. Predicting what language a website is in from the text on the website - multiclass classification.

In regression tasks, given a set of input variables, the goal is to predict a continuous number, or a floating-point number in programming terms.

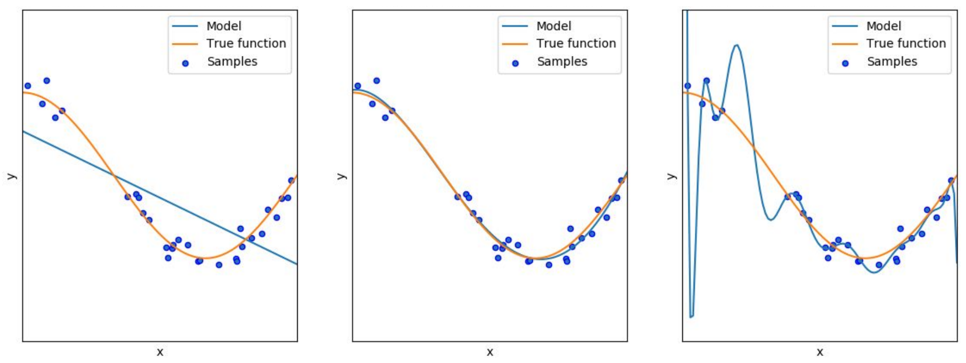
Eg. Predicting a person's annual income from their education, their age, and where they live is an example of a regression task. When predicting income, the predicted value is an amount, and can be any number in a given range.
Thus regression deals with outputs that are continuous values.

43. Discuss the terms Overfitting and Underfitting in model selection with an example.
In supervised learning, we want to build a model on the training data and then be able to make accurate predictions on new, unseen data that has the same characteristics as the training set that we used. If a model is able to make accurate predictions on unseen data, we say it is able to generalize from the training set to the test set. We want to build a model that is able to generalize as accurately as possible.

Building a model that is too complex for the amount of information we have, is called overfitting. Overfitting occurs when you fit a model too closely to the particularities of the training set and thus obtain a model that works well on the training set but is not able to generalize to new data.

On the other hand, if your model is too simple then you might not be able to capture all the aspects of and variability in the data, and your model will do badly on the training set and test set. Choosing too simple a model is called underfitting.

Example:



Case 1: Underfitting: Model passes straight through the training set with no regard for the data! This is because an underfit model has low variance and high bias. Variance refers to how much the model is dependent on the training data. For the case of a 1 degree polynomial, the model depends very little on the training data because it barely pays any attention to the points! Instead, the model has high bias, which means it makes a strong assumption about the data. For this example, the assumption is that the data is linear, which is evidently quite wrong. When the model makes test predictions, the bias leads it to make inaccurate estimates.

Case 2: Sweet spot between underfitting and overfitting. Model (2 degree polynomial) generalizes the data well.

Case 3: Overfitting: Model passes through every point in training set! This is because an overfit model has high variance. For the case of a 25 degree polynomial, the model depends a lot on the training data and varies greatly with changes in data!

44. Explain the k-NN algorithm with sample code.
The k-NN algorithm is arguably the simplest machine learning algorithm. Building the model consists only of storing the training dataset.

Algorithm:
Step 1: Determine parameter K = number of nearest neighbours
Step 2: Calculate the distance between new data point and all the training examples
Step 3: Sort the distances and determine the K nearest neighbours based on minimum distance.
Step 4: Gather the category Y of the K nearest neighbours
Step 5: Use simple majority of the category of K nearest neighbours as the prediction value for the new data point.

Sample Code:

Split data into training and test set so as to evaluate generalization performance:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test =
train_test_split(cancer.data, cancer.target,
stratify=cancer.target, random_state=0)
```

Import and instantiate the classifier. Set parameters i.e k = 3 (number of neighbours to use)

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
```

Fit the classifier using the training set. For KNeighborsClassifier this means storing the dataset, so as to compute neighbors during prediction:

```
clf.fit(X_train, y_train)
```

To make predictions on the test data, the predict method is called. For each data point in the test set, this computes its nearest neighbors in the training set and finds the most common class among these.

```
print("Test set predictions: {}".format(clf.predict(X_test)))
```

To evaluate how well the model generalizes, the score method is called with the test data together with the test labels:

```
print("Test set accuracy: {:.2f}".format(clf.score(X_test,
y_test)))
```

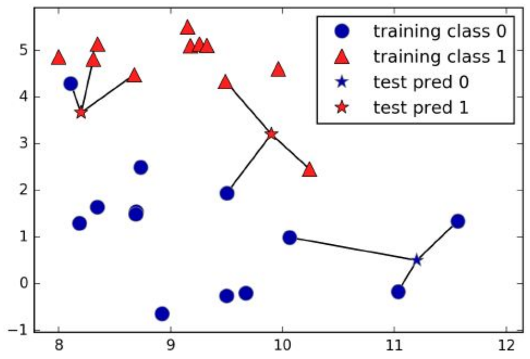
If the model is say about 86% accurate, it means the model predicted the class correctly for 86% of the samples in the test dataset.

45. Walkthrough k-NN classification algorithm with an example.

The k-NN algorithm is arguably the simplest machine learning algorithm. Building the model consists only of storing the training dataset.

Algorithm:
Step 1: Determine parameter K = number of nearest neighbours
Step 2: Calculate the distance between new data point and all the training examples
Step 3: Sort the distances and determine the K nearest neighbours based on minimum distance.
Step 4: Gather the category Y of the K nearest neighbours
Step 5: Use simple majority of the category of K nearest neighbours as the prediction value for the new data point.

Example:
In the example below, we can consider an arbitrary number of neighbors - K = 3. As we are considering more than one neighbor, we use voting to assign a label. This means that for each test point, we count how many neighbors out of the 3 closest belong to class 0 and how many neighbors belong to class 1. We then assign the class that is more frequent: in other words, the majority class among the k-nearest neighbors. For more classes, we count how many neighbors belong to each class and again predict the most common class.

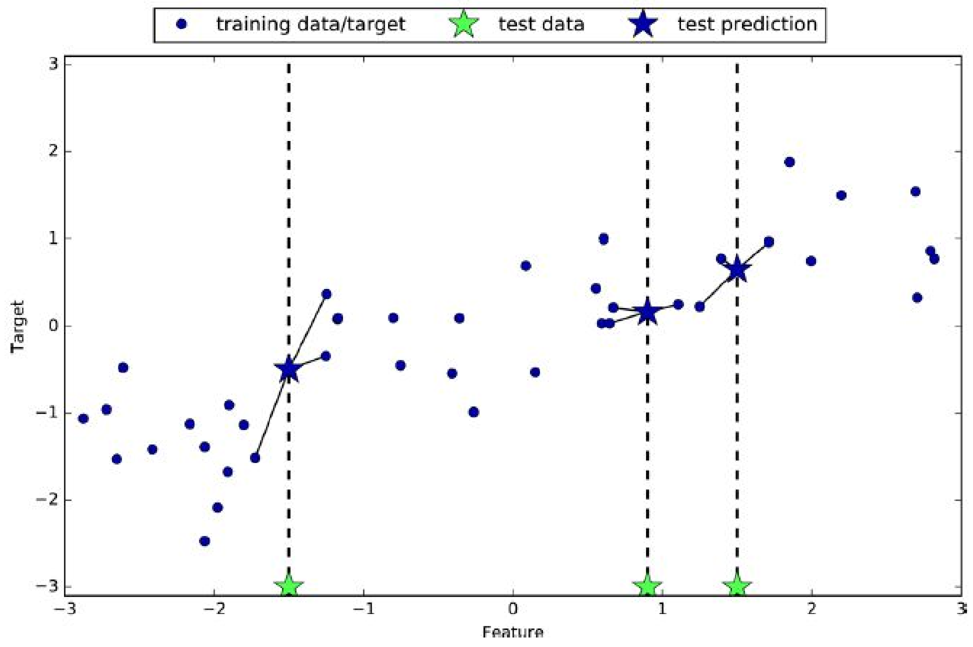


46. Explain the k-neighbourhood regression technique.

The k-neighbourhood regression technique is similar to the k-NN algorithm but is used for regression problems as opposed to classification.

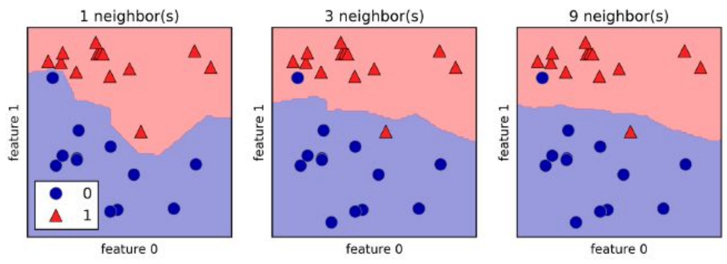
Algorithm:
Step 1: Determine parameter K = number of nearest neighbours
Step 2: Calculate the distance between new data point and all the training examples
Step 3: Sort the distances and determine the K nearest neighbours based on minimum distance.
Step 4: Gather the output value Y of the K nearest neighbours
Step 5: Use the average, or mean, of the K nearest neighbors as the prediction value for the new data point.

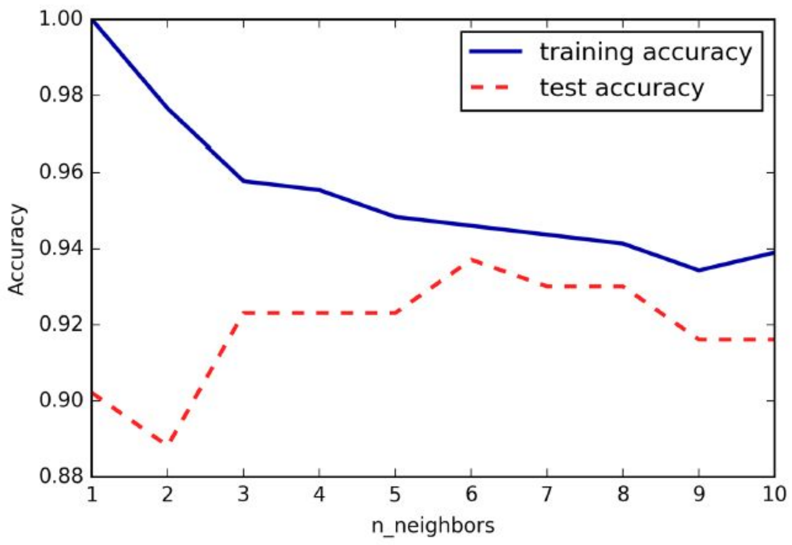
Example: Below is a plot of points with one feature on the X-axis and the target on the Y-axis. The three test data points are in green stars on the x-axis. The prediction using 3 neighbor is the mean of the target value of the nearest neighbor. The predictions are shown in blue stars.



47. How to calculate the effectiveness of k-NN algorithms?

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target,
random_state=66)
training_accuracy = []
test_accuracy = []
# try n_neighbors from 1 to 10
neighbors_settings = range(1, 11)
for n_neighbors in neighbors_settings:
# build the model
clf = KNeighborsClassifier(n_neighbors=n_neighbors)
clf.fit(X_train, y_train)
# record training set accuracy
training_accuracy.append(clf.score(X_train, y_train))
# record generalization accuracy
test_accuracy.append(clf.score(X_test, y_test))
plt.plot(neighbors_settings, training_accuracy,
label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test
accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
```





48. What is the biggest strength of linear regression model? What is its weakness?
Linear regression, or ordinary least squares (OLS), is the simplest and most classic linear method for regression. General formula:

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + ... + w[p] * x[p] + b$$

Linear regression finds the parameters w and b that minimize the mean squared error between predictions and the true regression targets, y, on the training set. The mean squared error is the sum of the squared differences between the predictions and the true values.

Strengths:

- Simple model, easy for beginners to understand
- No parameters to tune
- Useful as many problems can be *transformed* into linear regression problems

Weaknesses:

- Overfitting: Linear regression has no parameters, which is a benefit, but it also has no way to control model complexity. For a one-dimensional dataset, there is little danger of overfitting, as the model is very simple (or restricted). However, with higher-dimensional datasets, linear models become more powerful, and there is a higher chance of overfitting.
- Linear Regression Is Limited to Linear Relationships: By its nature, linear regression only looks at linear relationships between dependent and independent variables. That is, it assumes there is a straight-line relationship between them which may not always be true.
- Sensitive to outliers:

49. What is Ridge regression? How is it different from Linear regression models?
Ridge regression is also a linear model for regression, so the formula it uses to make predictions is the same one used for ordinary least squares i.e.

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + ... + w[p] * x[p] + b$$

However in ridge regression, the coefficients (w) are chosen not only so that they predict well on the training data, but also to fit an additional constraint i.e the magnitude of coefficients to be as small as possible. In other words, all entries of w should be close to zero. Intuitively, this means each feature should have as little effect on the outcome as possible (which translates to having a small slope), while still predicting well. This constraint is an example of what is called regularization. Regularization means explicitly restricting a model to avoid overfitting. The particular kind used by ridge regression is known as L2 regularization.

The Ridge model makes a trade-off between the simplicity of the model (near-zero coefficients) and its performance on the training set. How much importance the model places on simplicity versus training set performance can be specified by the user, using the alpha parameter. The optimum setting of alpha depends on the particular dataset being used. Increasing alpha forces coefficients to move more toward zero, which decreases training set performance but might help generalization.

It is seen through experiments that the training set score of Ridge is lower than for Linear Regression, while the test set score is higher. This is consistent with our expectation. With linear regression, overfitting is more likely. Ridge is a more restricted model, and so less likely to overfit the data.

50. Compare the relative advantage of Lasso over Linear and Ridge regression models.

An alternative to Ridge for regularizing linear regression is Lasso. The lasso model also restricts coefficients to be close to zero, but in a slightly different way, called L1 regularization as opposed to ridge which uses L2 regularization. The consequence of L1 regularization is that when using the lasso, some coefficients are exactly zero. This means some features are entirely ignored by the model. This can be seen as a form of automatic feature selection. Having some coefficients be exactly zero often makes a model easier to interpret, and can reveal the most important features of your model.

Similar to Ridge, the Lasso also has a regularization parameter, alpha, that controls how strongly coefficients are pushed toward zero. A lower alpha allowed us to fit a more complex model, which worked better on the training and test data. If we set alpha too low, however, we again remove the effect of regularization and end up overfitting, with a result similar to Linear Regression.

Thus Lasso has the advantage of preventing overfitting in a linear regression model as well as doing automatic feature selection

51. Write pseudocode to simulate various linear regression models.

Linear Regression

```
from sklearn.linear_model import LinearRegression
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=42)
lr = LinearRegression().fit(X_train, y_train)
print("lr.coef_: {}".format(lr.coef_))
print("lr.intercept_: {}".format(lr.intercept_))
print("Training set score: {:.2f}".format(lr.score(X_train,
y_train)))
```

```
print("Test set score: {:.2f}".format(lr.score(X_test,
y_test)))
```

Ridge Regression

```
from sklearn.linear_model import Ridge
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=42)
ridge = Ridge(alpha=1.0).fit(X_train, y_train)
print("ridge.coef_: {}".format(ridge.coef_))
print("ridge.intercept_: {}".format(ridge.intercept_))
print("Training set score:
{:.2f}".format(ridge.score(X_train, y_train)))
print("Test set score: {:.2f}".format(ridge.score(X_test,
y_test)))
```

Lasso Regression

```
from sklearn.linear_model import Lasso
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=42)
lasso = Lasso().fit(X_train, y_train)
print("lasso.coef_: {}".format(lasso.coef_))
print("lasso.intercept_: {}".format(lasso.intercept_))
print("Training set score:
{:.2f}".format(lasso.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lasso.score(X_test,
y_test)))
print("Number of features used: {}".format(np.sum(lasso.coef_
!= 0)))
```

52. How can linear models be used for multi-class classification?

Many linear classification models are used for binary classification only, and don't extend naturally to the multiclass case (with the exception of logistic regression). A common technique to extend a binary classification algorithm to a multiclass classification algorithm is the one-vs.-rest approach. In the one-vs.-rest approach, a binary model is learned for each class that tries to separate that class from all of the other classes, resulting in as many binary models as there are classes. To make a prediction, all binary classifiers are run on a test point.

The classifier that has the highest score on its single class "wins," and this class label is returned as the prediction.

Having one binary classifier per class results in having one vector of coefficients (w) and one intercept (b) for each class. The class for which the result of the classification confidence formula given here, is highest is the assigned class label:

w[0] * x[0] + w[1] * x[1] + ... + w[p] * x[p] + b

Let's apply the one-vs.-rest method to a simple three-class classification dataset. We use a two-dimensional dataset, where each class is given by data sampled from a Gaussian distribution

```
from sklearn.datasets import make_blobs
x, y = make_blobs(random_state=42)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
plt.legend(["Class 0", "Class 1", "Class 2"])
```

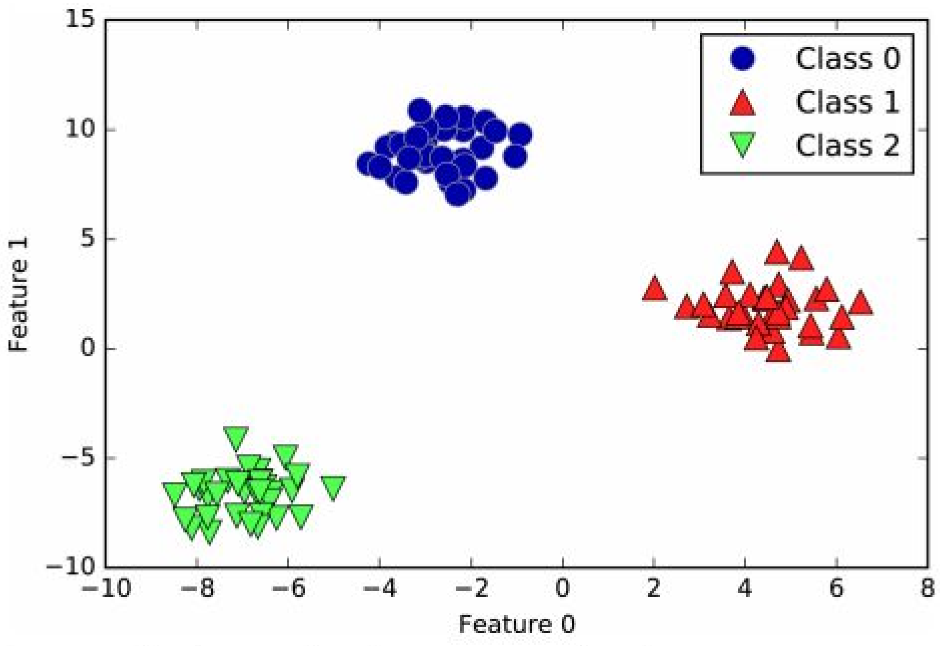


Figure 2-19. Two-dimensional toy dataset containing three classes

Now, we train a LinearSVC classifier on the dataset:

```
linear_svm = LinearSVC().fit(X, y)
```

```
print("Coefficient shape: ", linear_svm.coef_.shape)
print("Intercept shape: ", linear_svm.intercept_.shape)
```

Output :
Coefficient shape: (3, 2)
Intercept shape: (3,)

We see that the shape of the coef_ is (3, 2), meaning that each row of coef_ contains the coefficient vector for one of the three classes and each column holds the coefficient value for a specific feature (there are two in this dataset). The intercept_ is now a one-dimensional array, storing the intercepts for each class.

53. Write the pseudocode for Naive bayes classifiers.

```
import numpy as np
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
Y = np.array([1, 1, 1, 2, 2, 2])
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X, Y)
print(clf.predict([[-0.8, -1]]))
```

54. Why do Naive bayes classifiers perform better than linear models?

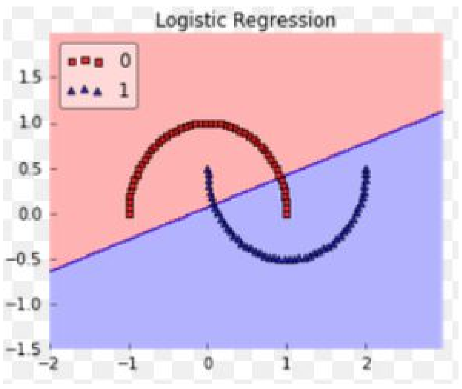
Naive Bayes classifiers tend to be faster in training compared to linear models. The reason that naive Bayes models are so efficient is that they learn parameters by looking at each feature individually and collect simple per-class statistics from each feature. The models work very well with high-dimensional sparse data and are relatively robust to the parameters. Naive Bayes models are great baseline models and are often used on very large datasets, where training even a linear model might take too long. The price paid for this efficiency is that naive Bayes models often provide generalization performance that is slightly worse than that of linear classifiers like LogisticRegression and LinearSVC.

55. Write the pseudocode for decision trees.

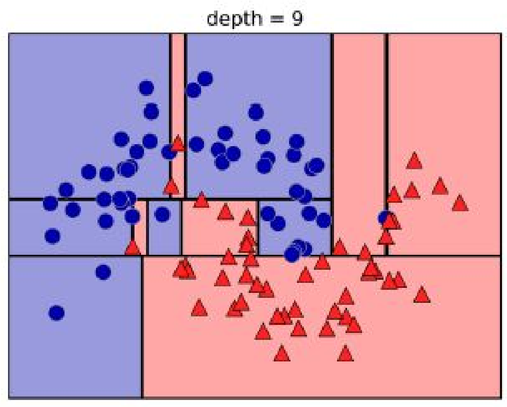
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target,
random_state=42)
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set:
{:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set:
{:.3f}".format(tree.score(X_test, y_test)))
```

56. Define a sample model for a decision tree and prove its advantage over binary classifiers.

More complex decision boundaries, easy to visualize decision making process, no scaling required. Example decision tree works better on half moon dataset



Binary Classifier



Decision Tree

57. How can the traversing of the tree be minimised using feature importance factor?

Instead of looking at the whole tree, which can be taxing, there are some useful properties that we can derive to summarize the workings of the tree. The most commonly used summary is feature importance, which rates how important each feature is for the decision a tree makes. It is a number between 0 and 1 for each feature, where 0 means "not used at all" and 1 means "perfectly predicts the target." The feature importances always sum to 1.

However, if a feature has a low feature importance value, it doesn't mean that this feature is uninformative. It only means that the feature was not picked by the tree, likely because another feature encodes the same information. In contrast to the coefficients in linear models, feature importances are always positive, and don't encode which class a feature is indicative of. In fact, there might not be such a simple relationship between features and class in some cases.

58. Write the pseudocode for random forests.

```
from sklearn.tree import RandomForestClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target,
random_state=42)
tree = RandomForestClassifier(n_estimators=100,
random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set:
{:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set:
{:.3f}".format(tree.score(X_test, y_test)))
```

59. Discuss the relative importance of random forests over decision trees.
A main drawback of decision trees is that they tend to overfit the training data. Random forests are one way to address this problem. A random forest is essentially a collection of decision trees, where each tree is slightly different from the others. The idea behind random forests is that each tree might do a relatively good job of predicting, but will likely overfit on part of the data. If we build many trees, all of which work well and overfit in different ways, we can reduce the amount of overfitting by averaging their results. This reduction in overfitting, while retaining the predictive power of the trees, can be shown using rigorous mathematics.

To implement this strategy, we need to build many decision trees. Each tree should do an acceptable job of predicting the target, and should also be different from the other trees. There are two ways in which the trees in a random forest are randomized: by selecting the data points used to build a tree and by selecting the features in each split test.

Random forest gives nonzero importance to many more features than the single tree. The randomness in building the random forest forces the algorithm to consider many possible explanations, the result being that the random forest captures a much broader picture of the data than a single tree.

Random forests for regression and classification are currently among the most widely used machine learning methods. They are very powerful, often work well without heavy tuning of the parameters, and don't require scaling of the data. Random forests usually work well even on very large datasets, and training can easily be parallelized over many CPU cores within a powerful computer.

60. Compare various supervised learning techniques and bring out their pros and cons.

KNN
One of the strengths of k-NN is that the model is very easy to understand, and often gives reasonable performance without a lot of adjustments. Using this algorithm is a good baseline method to try before considering more advanced techniques. Building the nearest neighbors model is usually very fast, but when your training set is very large (either in number of features or in number of samples) prediction can be slow. This approach often does not perform well on datasets with many features (hundreds or more), and it does particularly badly with datasets where most features are 0 most of the time (so-called sparse datasets).

Linear Models
Linear models are very fast to train, and also fast to predict. They scale to very large datasets and work well with sparse data. Another strength of linear models is that they make it relatively easy to understand how a prediction is made for regression and classification. Unfortunately, it is often not entirely clear why coefficients are the way they are. This is particularly true if your dataset has highly correlated features; in these cases, the coefficients might be hard to interpret.
Linear models often perform well when the number of features is large compared to the number of samples. They are also often used on very large datasets, simply because it's not feasible to train other models. However, in lower-dimensional spaces, other models might yield better generalization performance.

Naive Bayes Classifier
The naive Bayes models share many of the strengths and weaknesses of the linear models. They are very fast to train and to predict, and the training procedure is easy to understand.

The models work very well with high-dimensional sparse data and are relatively robust to the parameters. Naive Bayes models are great baseline models and are often used on very large datasets, where training even a linear model might take too long.

Decision Trees

Decision trees have two advantages over many of the algorithms: the resulting model can easily be visualized and understood by nonexperts (at least for smaller trees), and the algorithms are completely invariant to scaling of the data. As each feature is processed separately, and the possible splits of the data don’t depend on scaling, no preprocessing like normalization or standardization of features is needed for decision tree algorithms. In particular, decision trees work well when you have features that are on completely different scales, or a mix of binary and continuous features.

The main downside of decision trees is that even with the use of pre-pruning, they tend to overfit and provide poor generalization performance.

Random Forest

Random forests for regression and classification are currently among the most widely used machine learning methods. They are very powerful, often work well without heavy tuning of the parameters, and don’t require scaling of the data. Essentially, random forests share all of the benefits of decision trees, while making up for some of their deficiencies. While building random forests on large datasets might be somewhat time consuming, it can be parallelized across multiple CPU

Random forests don’t tend to perform well on very high dimensional, sparse data, such as text data. For this kind of data, linear models might be more appropriate. Random forests usually work well even on very large datasets, and training can easily be parallelized over many CPU cores within a powerful computer. However, random forests require more memory and are slower to train and to predict than linear models. If time and memory are important in an application, it might make sense to use a linear model instead.

UNIT 4

64. How to apply various data transformation techniques to clean the data?
Preprocessing methods like the scalers are usually applied before applying a supervised machine learning algorithm. As an example, say we want to apply the kernel SVM (SVC) to the cancer dataset, and use MinMaxScaler for preprocessing the data. We start by loading our dataset and splitting it into a training set and a test set (we need separate training and test sets to evaluate the supervised model we will build after the preprocessing):

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data,
cancer.target,
random_state=1)
print(X_train.shape)
print(X_test.shape)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
print("per-feature minimum after
scaling:\n{}".format(X_test_scaled.min(axis=0)))
print("per-feature maximum after
scaling:\n{}".format(X_test_scaled.max(axis=0)))
```

66. Explain the rationale behind PCA.

Transforming data using unsupervised learning can have many motivations. The most common motivations are visualization, compressing the data, and finding a representation that is more informative for further processing.
Principal component analysis is a method that rotates the dataset in a way such that the rotated features are statistically uncorrelated. This rotation is often followed by selecting only a subset of the new features, according to how important they are for explaining the data. PCA is sensitive to the relative scaling of the original variables.
The algorithm proceeds by first finding the direction of maximum variance, labeled "Component 1." This is the direction (or vector) in the data that contains most of the information, or in other words, the direction along which the features are most correlated with each other.