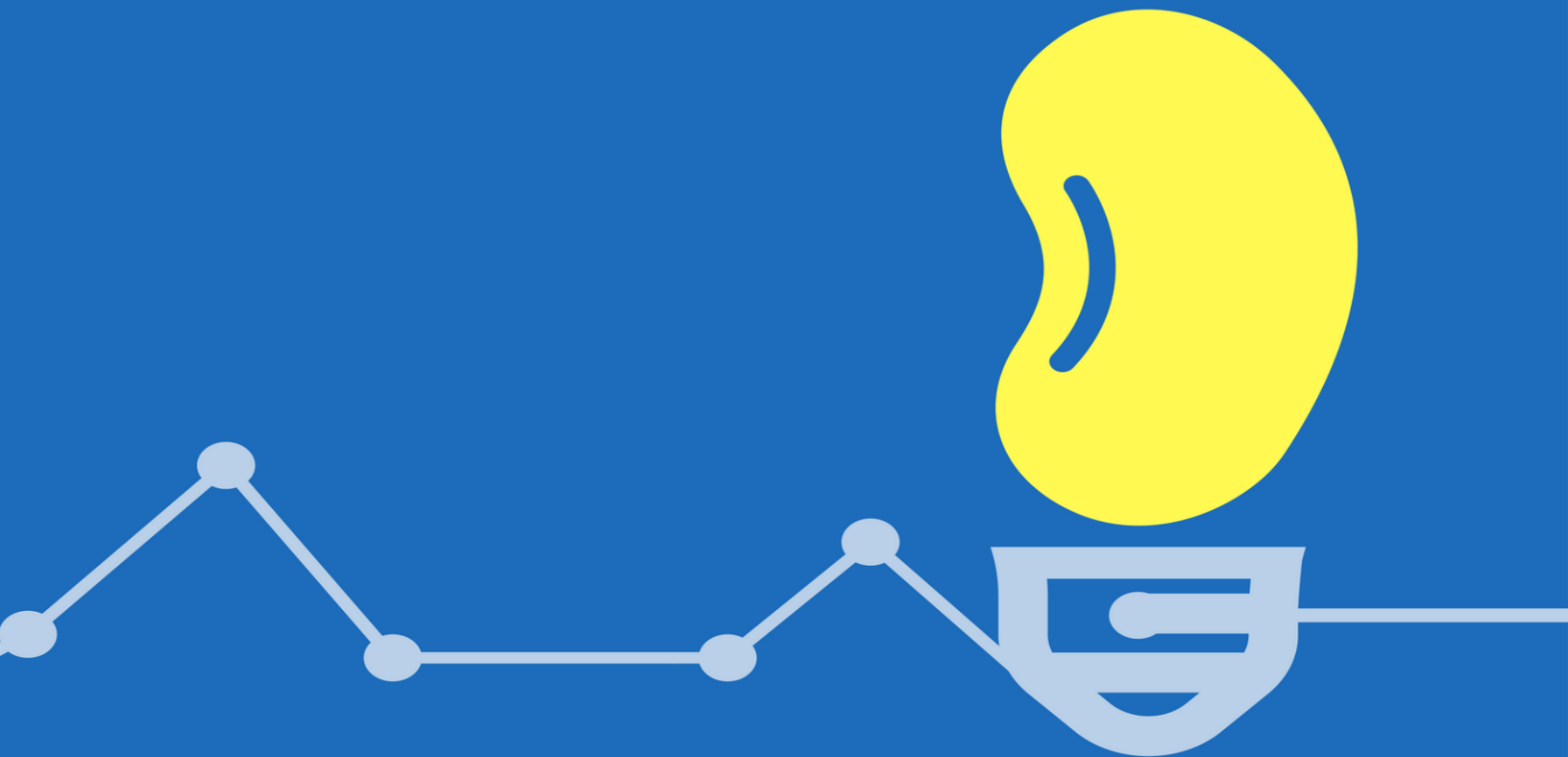


NUMSENSE!

Data Science Tutorials for the Layman

(No math added.)



Annalyn Ng & Kenneth Soo

Numsense! Data Science for the Layman

No Math Added

Annalyn Ng and Kenneth Soo

This book is for sale at <http://leanpub.com/numsense>

This version was published on 2016-10-19

ISBN 978-981-11-1068-9



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2016 Annalyn Ng and Kenneth Soo

Tweet This Book!

Please help Annalyn Ng and Kenneth Soo by spreading the word about this book on [Twitter](#)!

The suggested tweet for this book is:

[Layman data science tutorials in a free e-Book!](#)

The suggested hashtag for this book is [#algobeans](#).

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

<https://twitter.com/search?q=#algobeans>

Numsense! is dedicated to you by two data science enthusiasts, Annalyn Ng (University of Cambridge) and Kenneth Soo (Stanford University).

We noticed that while data science is increasingly used to improve workplace decisions, many people know little about the field. Hence, we compiled these tutorials so that everyone and anyone can learn - be it an aspiring student or enterprising business professional.

Each tutorial covers the important functions and assumptions of a data science technique, without any math or jargon. We also illustrate these techniques with real-world data and examples.

Hope you like it!

Annalyn & Kenneth

algobeans.com

Contents

Fitting a Predictive Model	1
Uncovering Trends to Make Predictions	1
Parameter Tuning	2
Validation	3
Regularization	4
Conclusion	5
<i>k</i>-Means Clustering	6
Finding Customer Clusters	6
Example: Personalities of Movie Fans	6
Defining Clusters	8
Limitations	11
Principal Component Analysis	12
Exploring Nutritional Content of Food	12
Principal Components	12
Example: Analyzing Food Groups	14
Limitations	18
Association Rules	21
Discovering Purchasing Patterns	21
Support, Confidence and Lift	21
Example: Mapping Grocery Transactions	23
Apriori Algorithm	25
Limitations	26
Correlation and Regression	27
Selecting People to Recruit	27
Example: Profiling Military Commanders	27
Deriving a Trend Line	28
Limitations	30
<i>k</i>-Nearest Neighbors and Anomaly Detection	32
Identifying Categories by Chemical Make-Up	32
Example: Distilling Wine Differences	32

CONTENTS

Classification Based on Neighbors	34
Anomaly Detection	36
Limitations	36
Support Vector Machine	38
<i>“No” or “Oh no”?</i>	38
Example: Predicting Heart Disease	38
Delineating an Optimal Boundary	40
Limitations	41
Decision Tree	43
Predicting Survival in a Disaster	43
Example: Surviving the Titanic	44
Generating a Decision Tree	45
Limitations	46
Random Forest	48
Wisdom of the Crowd	48
Example: Forecasting Crime	48
Ensembles	52
Bootstrap Aggregating (Bagging)	53
Limitations	54
A/B Testing and Multi-Armed Bandits	55
Basics of A/B testing	55
Limitations of A/B testing	55
Epsilon-Decreasing Strategy	56
Example: Multi-Arm Bandits	56
Limitations of an Epsilon-Decreasing Strategy	58
Fun Fact: Sticking to the Winner	59

Fitting a Predictive Model

Uncovering Trends to Make Predictions

Imagine a girl passing by a garden. She stops at a magnificent thicket adorned with blue and orange flowers. Peering through the flowers and leaves are several flower buds. Curious as to which color these buds would eventually bloom into, the girl starts to observe patterns in how the blue and orange flowers are distributed. Based on her observations, she is able to make guesses.

What the girl did is akin to the general task of prediction models. First, patterns are uncovered. Then, these patterns are used to make predictions. Patterns can be ambiguous, however. If another girl had tried the same task, she might have drawn different boundaries to separate the blue from orange flowers. Similarly, different prediction models might infer different patterns from the same set of data, and this might lead to different predictions. Figure 1 shows three scenarios of how we might infer distribution patterns for the blue and orange flowers.

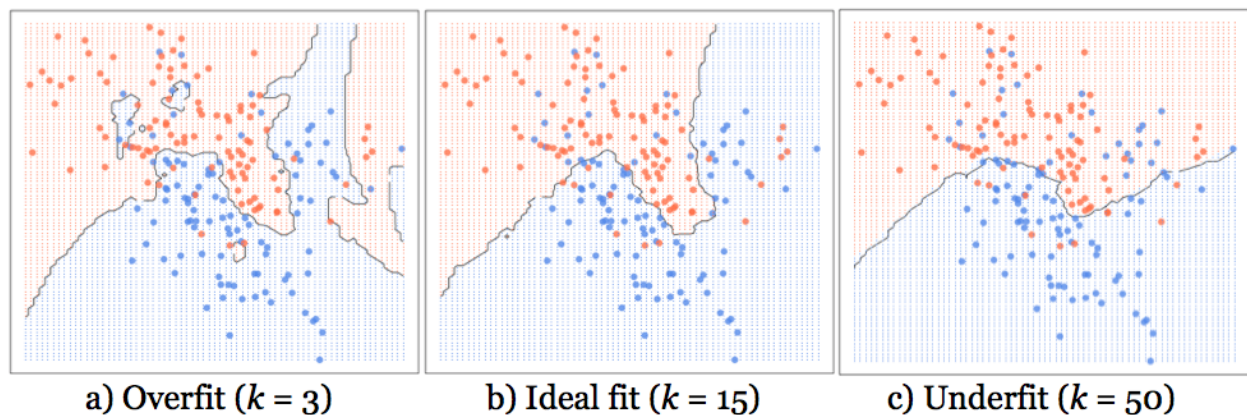


Figure 1. Comparing different ways to separate blue from orange flowers.

In Figure 1a, boundaries separating the two colors are too detailed. Random variations in flower growth have been mistaken as persistent patterns, which would not be generalizable to new flowers. This problem is called *overfitting*.

In Figure 1c, boundaries separating the two colors are too vague. Subtle patterns are overlooked, resulting in less information to guide predictions. This problem is called *underfitting*.

If a predictive model is overfitted or underfitted, its predictions might be inaccurate, undermining its original purpose. What we need is a predictive model like the one in Figure 1b - an ideal fit that strikes a balance between describing the current distribution of flowers, and being able to predict where new flowers would grow.

To derive a predictive model that fits our data well, we need to do two things:

Derive multiple models. We could invite multiple girls to draw different boundaries to separate the flowers, or we could simply ask the same girl to do so. Analogously, we could use different prediction algorithms, or simply tweak parts of a single algorithm to get different models. The latter technique is what we will cover, and it is called (*parameter tuning*).

Select the best model. After we have multiple boundaries drawn, we need to see which boundary is best at predicting colors of new flowers. An ideal model not only fits known data but also generalizes to unknown data. Assessing how well models predict new data is a process known as (*validation*).

Parameter Tuning

An algorithm is composed of parts called parameters, which can be tuned to make the algorithm work differently. A predictive model can be changed by tweaking its parameters. Table 1 lists examples of common algorithms which will be covered in this book, as well as their parameters.

Table 1. Examples of common tuning parameters.

Algorithm	Parameters
<i>k</i> -Nearest Neighbors	Number of nearest neighbors
Support Vector Machine	Soft margin constant Kernel parameters Insensitivity parameter
Decision Tree	Minimum size of terminal nodes Maximum number of terminal nodes Maximum depth of tree
Random Forest	All parameters of a decision tree Number of trees Number of variables to select at each split
Artificial Neural Network	Number of hidden layers Number of neurons in each layer Learning rate (i.e. step size) Number of training iterations Initial weights in each neuron

For instance, let us look at one of the simplest techniques for classifying objects, called *k*-nearest neighbors (*k*-NN). Suppose we need to classify new data points as either blue or orange. *k*-NN works on the idea that a data point would likely have the same color as other data points closest to

it. For example, if a data point is surrounded by 4 blue points and 1 orange point, that data point is likely a blue point by majority vote (see Figure 2).

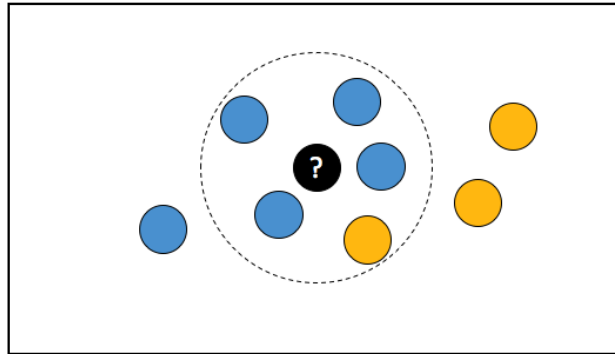


Figure 2. A data point is classified by majority votes from its 5 nearest neighbors.

The k in k -NN is a parameter that refers to the number of nearest neighbors to include in the majority voting process. In the above example, k equals to 5. Choosing the right value of k is a process called *parameter tuning*, and is critical to prediction accuracy.

If k is too small (Figure 1a), data points would match immediate neighbors only, amplifying errors. If k is too big (Figure 1c), data points would try to match far flung neighbors, diluting underlying patterns. But when k is just right (Figure 1b), data points would reference a suitable number of neighbors such that errors cancel out to reveal overall trends in the data, which could be used to make predictions.

While we can tune k based on current data, how can we assess if it would give accurate predictions for new data?

Validation

Validation is an assessment of how accurate a model is in predicting new data. However, instead of waiting for new flowers to bloom to assess our model, we could make a sketch of the current flowers, and randomly assign them into two groups - the first group of flowers, called the *training dataset*, would be used to generate and tune a prediction model, while the second group of flowers, called the *test dataset*, would be used to test the model's prediction accuracy. The model which predicts the correct color for the most number of flowers in the test dataset is taken to have the best parameter k . For the validation process to be effective, flowers should be randomly assigned into the training and test datasets, and should not be concentrated at any area on the thicket.

If the original dataset is small, we may not be able to set aside a test dataset without sacrificing accuracy from having less data to learn patterns from. Hence, instead of using two separate datasets for training and testing, there is a technique which lets us use one dataset for both purposes.

Cross-Validation

Cross-validation is a technique that maximizes the availability of data for validating a predictive model. It does this by dividing the dataset into several segments to test the model repeatedly.

In one iteration, all but one of the segments are used to train a predictive model of a desired parameter value. For instance, if we think that the optimal value for k is 5, we can use the 5 nearest neighbors in the training segments to predict the color for each flower in the last test segment. This process is repeated until each segment is used as the test segment exactly once.

As different segments are used to generate color predictions for each iteration, the resulting predictions would fluctuate. Simulating these variations in predictions results in a more robust estimate of actual predictive ability. The final estimate of the model's prediction accuracy is taken as the average of that across all the iterations (see Figure 3).

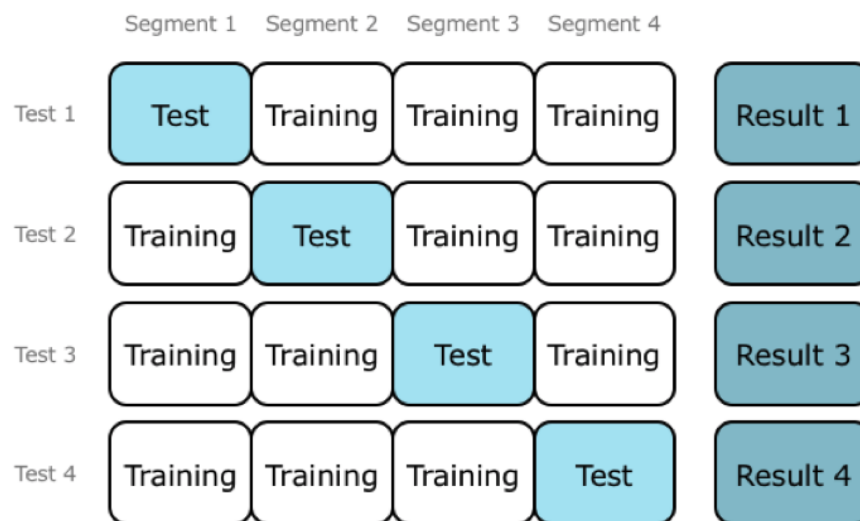


Figure 3. Cross-validating a dataset by dividing it into 4 segments. The final prediction accuracy is the average of the 4 results.

If results from cross-validation suggest that our model's prediction accuracy is low, we can go back to re-tune the parameters (e.g. value of k), or re-examine if our data is biased.

Regularization

As the complexity of a model's parameters increases, so too does the overall complexity of the model itself. This is hard to illustrate with k -NN as it only has one parameter. Instead, we can use [regression analysis](#) as an example.

To make a prediction with regression analysis, a trend line is derived based on predictor variables. For instance, if we would like to predict a person's income, we might use age as a predictor. A plausible trend line might then be a straight upward slope to indicate increasing income as age

increases. However, we could add more predictors into the model, such as IQ scores and parents' income. As more predictors are added, the model would grow more complex. The resulting trend might consist of more intricate curves and bumps rather than a simple straight line, thus inflating the risk of overfitting.

While we can select predictors to include in a regression, we are unable to determine how much weight a predictor should contribute to deriving the trend line. Instead, the regression algorithm computes the optimal predictor weights to derive a line that gives the lowest prediction error on the training dataset. So unlike the case for k -NN, we are sometimes unable to tune a model's parameters, as they are governed by its algorithm.

While a regression trend line might predict training data accurately, a complex trend is prone to overfitting. To keep a model's complexity in check, a penalty could be imposed on the model, which increases with the complexity of the model's parameters. Note that the value of the penalty is itself a tuning parameter. So in other words, we can introduce a tuning parameter to control parameters that cannot be tuned. This process is called *regularization*. By keeping a model's complexity in check, we help to maintain its predictive power.

Conclusion

In this chapter, we covered the fundamentals of how to fit a prediction model, which form the basis for all the techniques we are about to learn in this book. No matter which prediction technique we use, the processes of parameter tuning, validation and regularization are crucial in ensuring that our models would be effective in predicting new data.

***k*-Means Clustering**

Finding Customer Clusters

There are patterns in people's preferences. Take someone who likes the movie *50 First Dates* for instance. It is highly probable that they would like similar chick flicks such as *27 Dresses*. Grouping people by their preferences could help stores better target product advertisements to interested customers.

However, identifying customer groups is tricky. We do not know 1) how customers should be grouped, nor 2) how many groups exist. To answer these questions, we will explore a technique called *k-means clustering*. This technique can be used to cluster customers or products, where *k* represents the number of clusters identified, with each cluster being as distinct from other cluster as possible.

While this chapter uses the retail sector as an example, clustering is employed in a wide range of fields. For example, clustering can help to identify biological genotypes, or to pinpoint hot spots of criminal activity.

Example: Personalities of Movie Fans

To identify customer clusters, *k*-means clustering needs quantifiable customer information. A common one used is income — higher-income groups tend to purchase more branded products compared to lower-income groups. Hence, stores can use this information to limit advertisement of expensive products to higher-income groups.

Apart from income, there are other ways customers can be grouped. In this example, we grouped customers based on their personality traits, and used results to predict which movies they would like.

Personality data was obtained from Facebook users who completed a personality questionnaire. There were 4 personality traits considered: *extraversion* (how much you like social interactions), *conscientiousness* (how hardworking you are), *anxiousness* (how easily you get stressed) and *openness to experiences* (how receptive you are to novelty).

While there were 4 personality traits, customer groups are best visualized on a 2-dimensional plot. Hence, personality traits were paired and scores were summed within each pair — conscientiousness + extraversion, and anxiousness + openness to experience.

The 2 aggregated personality scores for each person were then used to predict which movie pages they had “*liked*” on Facebook, allowing us to define groups of movie fans by their distinct personality profiles (see Figure 1).

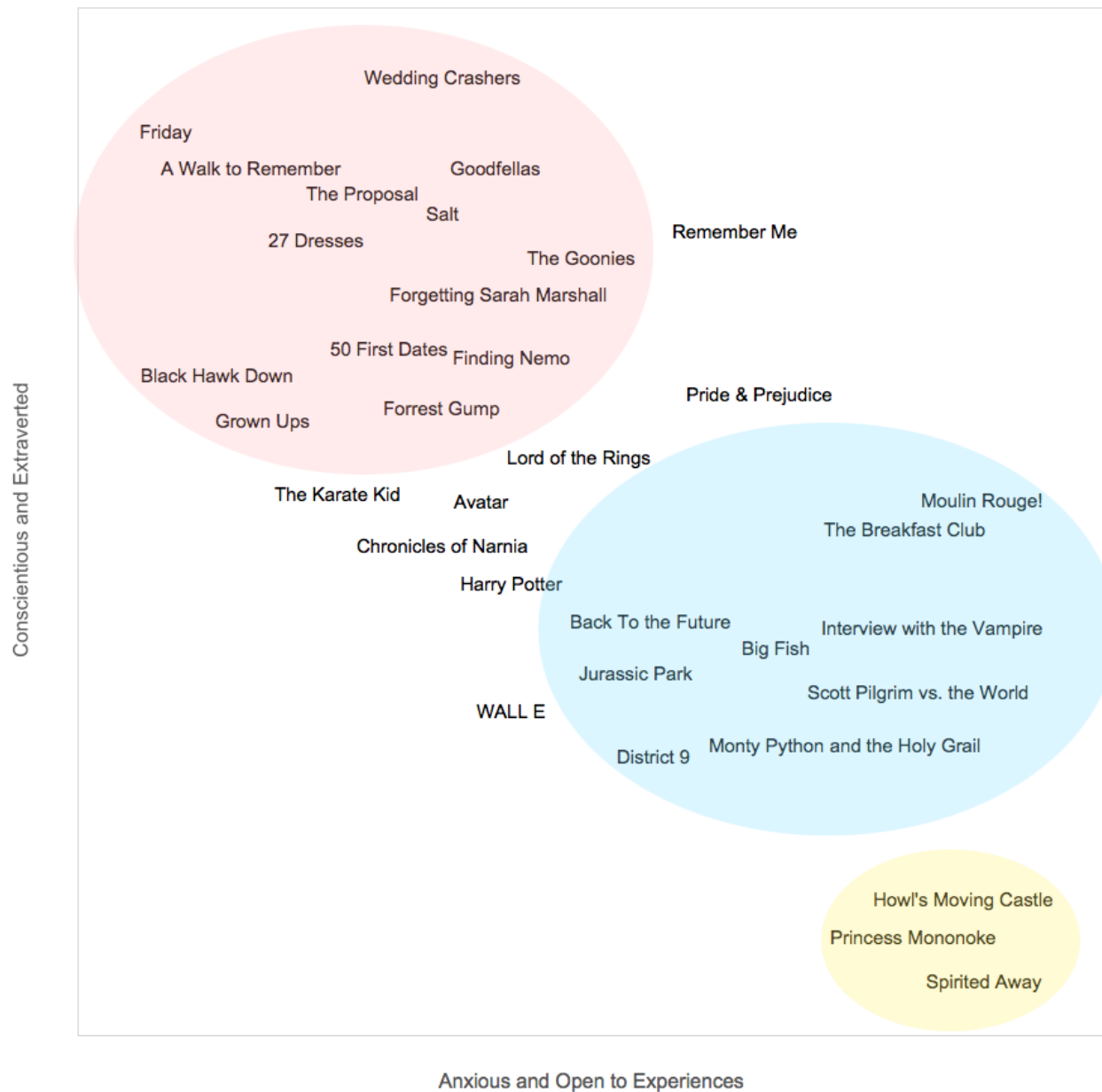


Figure 1. Personality profiles of movie fans.

In Figure 1, there appears to be three clusters:

- **Red:** Conscientious extraverts who like action and romance genres
- **Blue:** Anxious and open people who like avant-garde and fantasy genres
- **Yellow:** Introverts with social anxieties who like Japanese *anime* (“otaku” culture)
- Movies in the center seem to be general household favorites.

With this information, advertisements can be more targeted. If a person likes *50 First Dates*, a storekeep can recommend another movie in the same cluster, such as *27 Dresses*. Besides product

recommendation, identifying clusters also allow the storekeeper to bundle similar products for effective discounts.

Defining Clusters

In defining clusters, two questions have to be answered:

- How many clusters exist?
- What is the membership of each cluster?

How many clusters exist?

This is a subjective decision. While the movies in Figure 1 were grouped in 3 clusters, these clusters could be further broken down into smaller clusters. For instance, the blue cluster could have sub-clusters of a drama genre (comprising *Moulin Rouge* and *The Breakfast Club*) and a fantasy genre (comprising *Back to the Future* and *Jurassic Park*).

As the number of clusters increases, members within each clusters become more similar to each other, but at the same time, neighboring clusters also become less distinct from each other. In the extreme case, each data point could be a cluster on its own, which would be totally uninformative.

A balance needs to be struck. The number of clusters should be large enough so that meaningful patterns could be extracted to inform business decisions, yet small enough so that clusters remain distinct from each other.

One way to determine an appropriate number of clusters is to use a scree plot (see Figure 2).

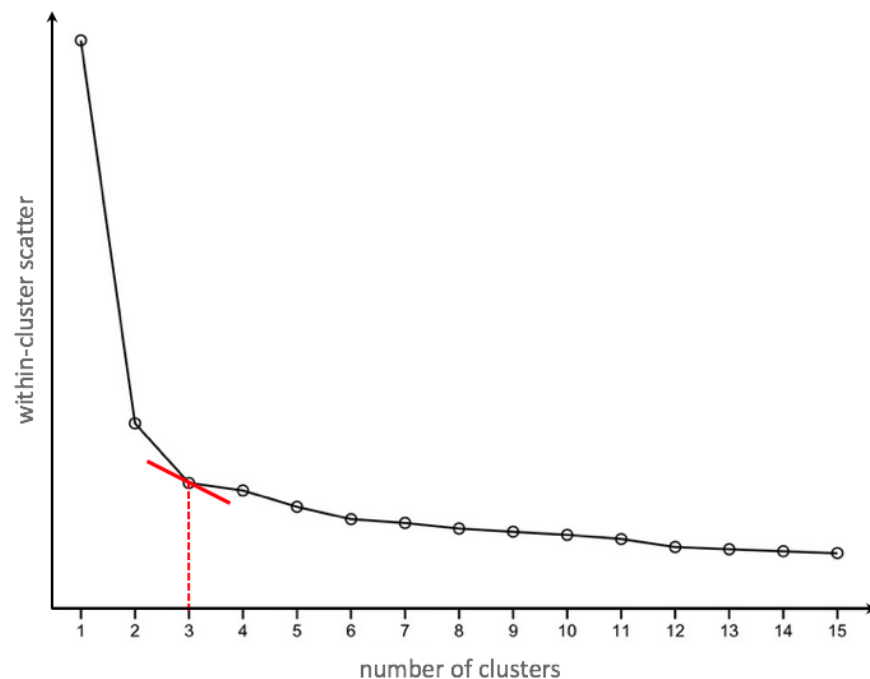


Figure 2. Scree plot showing a 'kink' where the number of clusters equals to 3.

A scree plot shows how the dissimilarity between cluster members (i.e. within-cluster scatter) decreases as the number of clusters increases. If there is only one cluster to which all members belong to, within-cluster scatter would be at its maximum. As the number of clusters increases, clusters grow more compact and cluster members become more homogenous.

An optimal number of clusters is suggested by the *kink* in a scree plot. In Figure 2, the kink is where the number of clusters equals to 3. At this point, the number of clusters derived can reduce within-cluster scatter to a reasonable degree, beyond which having any more clusters would yield increasingly smaller clusters that would be less distinguishable from each other. This is called the principle of *diminishing marginal returns*.

What is the membership of each cluster?

After determining a suitable number of clusters, we can determine the membership of each cluster.

A good cluster would comprise data points packed closely together. Hence, to check the validity of a cluster, we could verify how far its members are from their cluster's center. If a data point is far from its assigned cluster's center and closer to that of a neighboring cluster, its membership would be re-assigned.

However, the positions of cluster centers would be unknown initially, so they are approximated. Data points would then be assigned to cluster centers which they are closest to. Next, cluster centers would be re-positioned to actually be in the center of their members. Following which, cluster membership would be re-assigned again based on distance. This iterative process is summarized with a 2-cluster example illustrated in Figure 3.

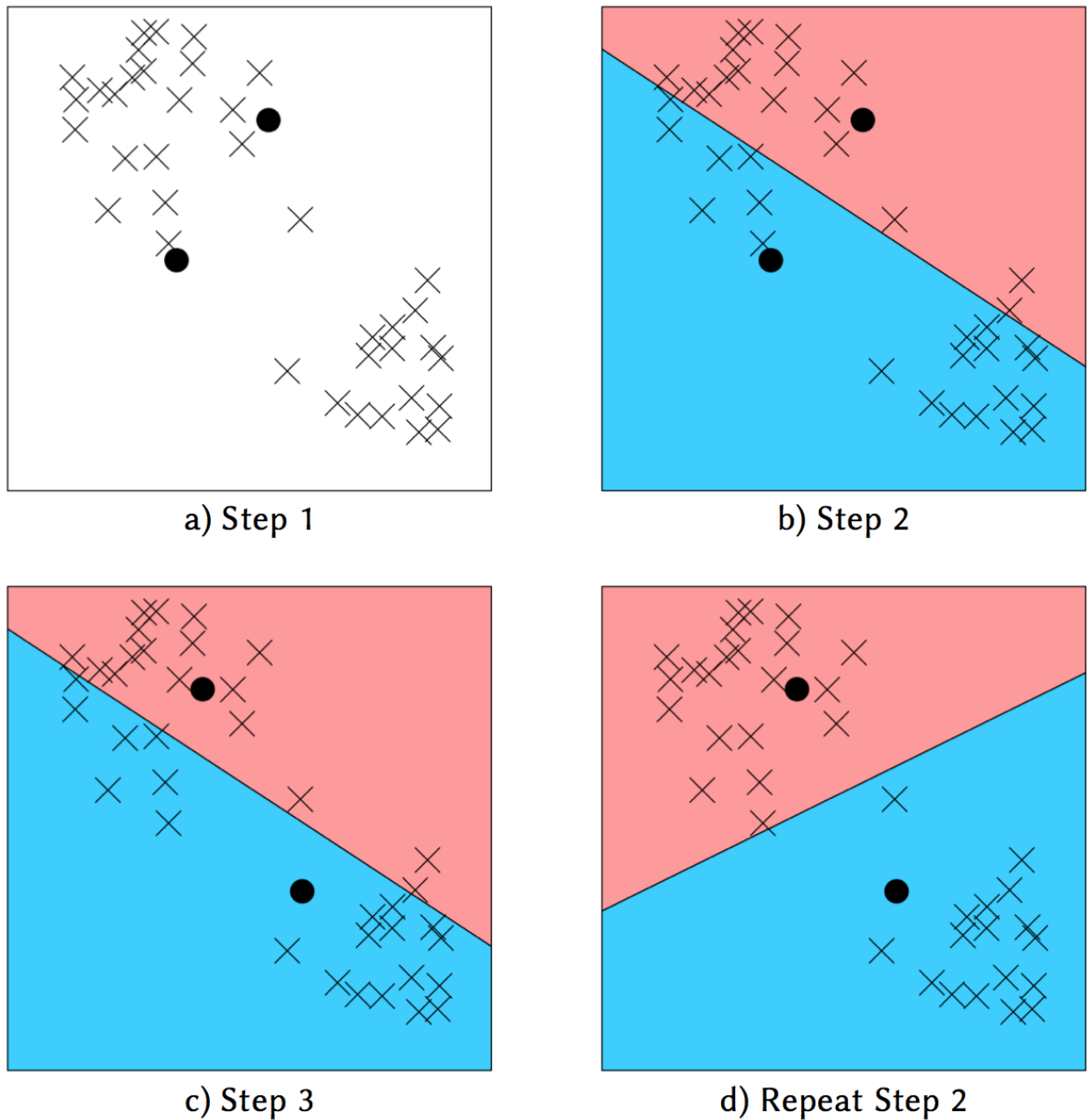


Figure 3. Iterative process in k -means clustering.

Step 1: Start by guessing where are the central points of each cluster. Let's call these pseudo-centers, since we do not yet know if they are actually at the center of their clusters.

Step 2: Assign each data point to the nearest pseudo-center. By doing so, we have just formed 2 clusters, red and blue.

Step 3: Update the location of pseudo-centers to be in the center of their respective members.

Step 4: Repeat the steps of re-assigning cluster members (Step 2) and re-positioning cluster centers

(Step 3), until there are no more changes to cluster membership. These 4 steps wrap up the process of determining cluster membership. The same process is used for 3 or more clusters, such as in Figure 1.

Despite our focus on 2-dimension analysis, clustering can also be done in 3 or more dimensions. Additional dimensions a storekeeper might want to consider could include a customer's age or frequency of visits. While difficult to visualize, we can still rely on computing programs to calculate distances between data points and cluster centers.

Limitations

Although *k*-means clustering is a useful tool, it is not without limitations:

Each data point can only be assigned to one cluster. Sometimes a data point might be in the middle of 2 clusters, having an equal chance of being assigned to either.

Clusters are assumed to be spherical. The iterative process of finding data points closest to a cluster center is akin to narrowing the cluster's radius, so that the resulting cluster is a compact sphere. This might pose a problem if the shape of an actual cluster is, for instance, an ellipse. An elongated cluster might be truncated, and its members subsumed into a nearby cluster.

Clusters are assumed to be discrete. *k*-means clustering does not permit clusters to overlap, nor to be nested within each other.

Instead of coercing each data point into one cluster, more robust clustering techniques compute probability values indicating how likely each data point might belong to each cluster. This would help identify non-spherical or overlapping clusters.

Despite the availability of alternative clustering techniques, the strength of the *k*-means clustering algorithm lies in its elegant simplicity. A good strategy might be to start with *k*-means clustering to get a basic understanding of the data structure, before diving into more advanced methods to examine areas where *k*-means clustering falls short.

References

Kosinski, M., Matz, S., Gosling, S., Popov, V. & Stillwell, D. (2015) *Facebook as a Social Science Research Tool: Opportunities, Challenges, Ethical Considerations and Practical Guidelines*. *American Psychologist*.

Principal Component Analysis

Exploring Nutritional Content of Food

Imagine that you are a nutritionist trying to explore the nutritional content of food. What is the best way to differentiate food items? By vitamin content? Protein levels? Or perhaps a combination of both?

Knowing the variables that best differentiate your items has several uses:

- **Visualization.** Using the right variables to plot items will give more insights.
- **Uncovering Clusters.** With good visualizations, hidden categories or clusters could be identified. Among food items for instance, we may identify broad categories like meat and vegetables, as well as sub-categories such as types of vegetables.

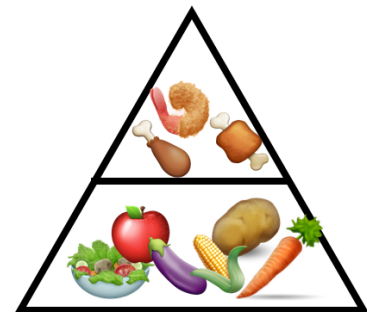


Figure 1. A simplified food pyramid.

The question is, how do we derive the variables that best differentiate items?

Principal Components

Principal Component Analysis (PCA) is a technique that finds underlying variables (known as *principal components*) that best differentiate your data points. Principal components are dimensions along which your data points are most spread out (see Figure 2).

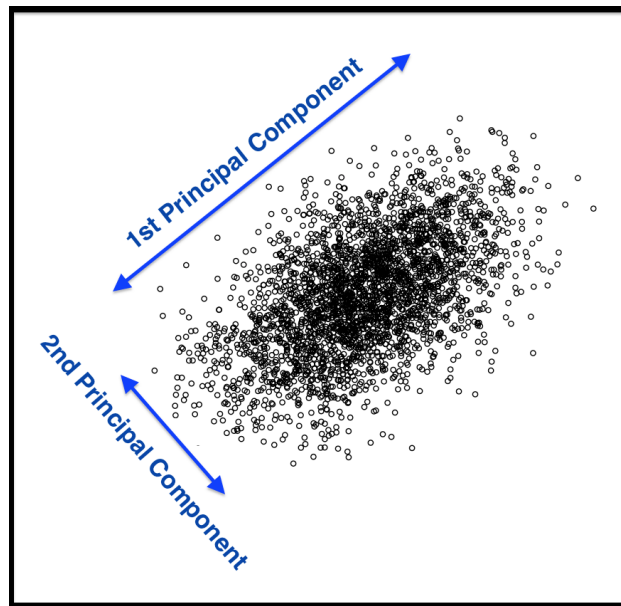


Figure 2. Visual representation of principal components.

A principal component can be expressed by one or more existing variables. For example, we may use a single variable – vitamin C – to differentiate food items. Because vitamin C is present in vegetables but absent in meat, the resulting plot (leftmost column in Figure 3) will differentiate vegetables from meat, but meat items will be clumped together.

To spread the meat items out, we can use fat content in addition to vitamin C levels, since fat is present in meat but absent in vegetables. However, fat and vitamin C levels are measured in different units. To combine the two variables, we first have to *normalize* them, i.e. shift them onto a uniform standard scale, which would allow us to calculate a new variable – *vitamin C minus fat*. As vitamin C spreads the vegetables upwards, we minus fat to spread the meats in the opposite direction downwards. Combining the two variables helps to spread out both vegetable and meat items (center column in Figure 3).

The spread can be further improved by adding fiber, of which vegetable items have varying levels. This new variable – *(vitamin C + fiber) minus fat* – achieves the best data spread yet (rightmost column in Figure 3).



Figure 3. How combinations of variables differentiate food items.

While in this demonstration we tried to derive principal components by trial-and-error, PCA does this by systematic computation.

Example: Analyzing Food Groups

Using open data from the *United States Department of Agriculture*, we analyzed the nutritional content of a random sample of food items. Four nutrition variables were analyzed: vitamin C, fiber, fat and protein. For fair comparison, food items were raw and measured by 100g.

Among food items, the presence of certain nutrients appear correlated. This is illustrated in the barplot in Figure 4 with 4 example items.

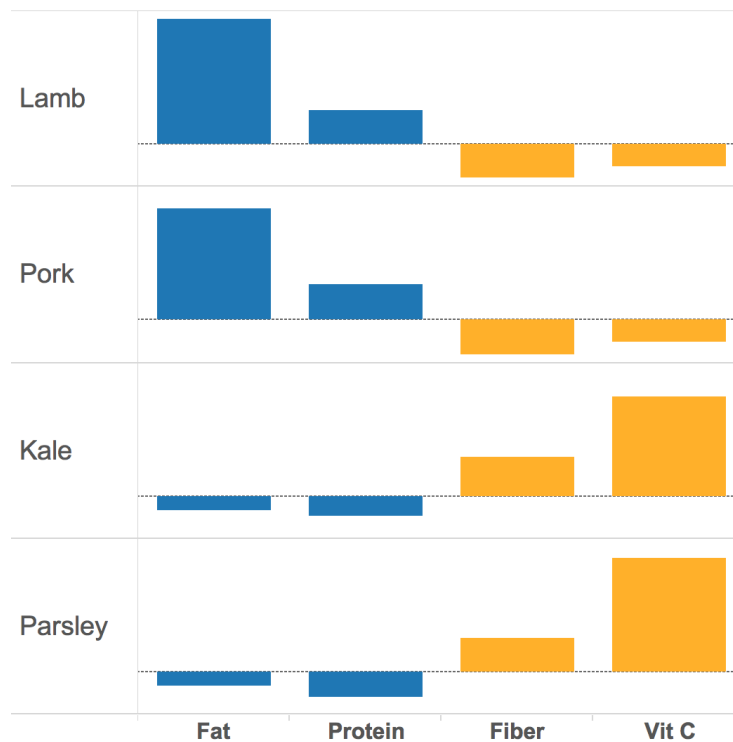


Figure 4. Comparing nutrition levels in different food items.

Specifically, fat and protein levels seem to move in the same direction with each other, and in the opposite direction from fiber and vitamin C levels. To confirm our hypothesis, we can check for [correlations](#) between the nutrition variables. As expected, there are significant positive correlations between fat and protein levels ($r = 0.56$), as well as between fiber and vitamin C levels ($r = 0.57$).

Therefore, instead of analyzing all 4 nutrition variables, we can combine highly-correlated variables, leaving just 2 dimensions to consider. This is the same strategy used in PCA – it examines correlations between variables to reduce the number of dimensions in the dataset. This is why PCA is called a *dimension reduction* technique.

Applying PCA to this food dataset results in the principal components displayed in Figure 5.

	PC1	PC2	PC3	PC4
Fat	-0.45	0.66	0.58	0.18
Protein	-0.55	0.21	-0.46	-0.67
Fiber	0.55	0.19	0.43	-0.69
Vitamin C	0.44	0.70	-0.52	0.22

Figure 5. How nutrition variables can be combined into principal components.

The numbers represent weights used in combining variables to derive principal components. So

instead of combining variables via trial-and-error like in Figure 1, PCA systematically computes the optimal combinations of variables that best differentiate our items.

For example, to get the top principal component (PC1) value for a particular food item, we add up the amount of fiber and vitamin C it contains, with slightly more emphasis on fiber, and then from that we subtract the amount of fat and protein it contains, with protein negated to a larger extent.

We observe that the top principal component (PC1) summarizes our findings so far – it has paired fat with protein, and fiber with vitamin C. It also takes into account the inverse relationship between the pairs. Hence, PC1 likely serves to differentiate meat from vegetables. The second principal component (PC2) is a combination of two unrelated nutrition variables – fat and vitamin C. It serves to further differentiate sub-categories within meat (using fat) and vegetables (using vitamin C).

Using the top 2 principal components to plot food items results in the best data spread thus far (shown in Figure 6).

Meat items (*blue*) have low PC1 values, and are thus concentrated on the left of the plot, on the opposite side from vegetable items (*orange*). Among meats, seafood items (*dark blue*) have lower fat content, so they have lower PC2 values and are at the bottom of the plot. Several non-leafy vegetarian items (*dark orange*), having lower vitamin C content, also have lower PC2 values and appear at the bottom.

Choosing the Number of Components. As principal components are derived from existing variables, the information available to differentiate data points is constrained by the number of variables you start with. Hence, the above PCA on food items generated 4 principal components, corresponding to the original number of variables in the dataset.

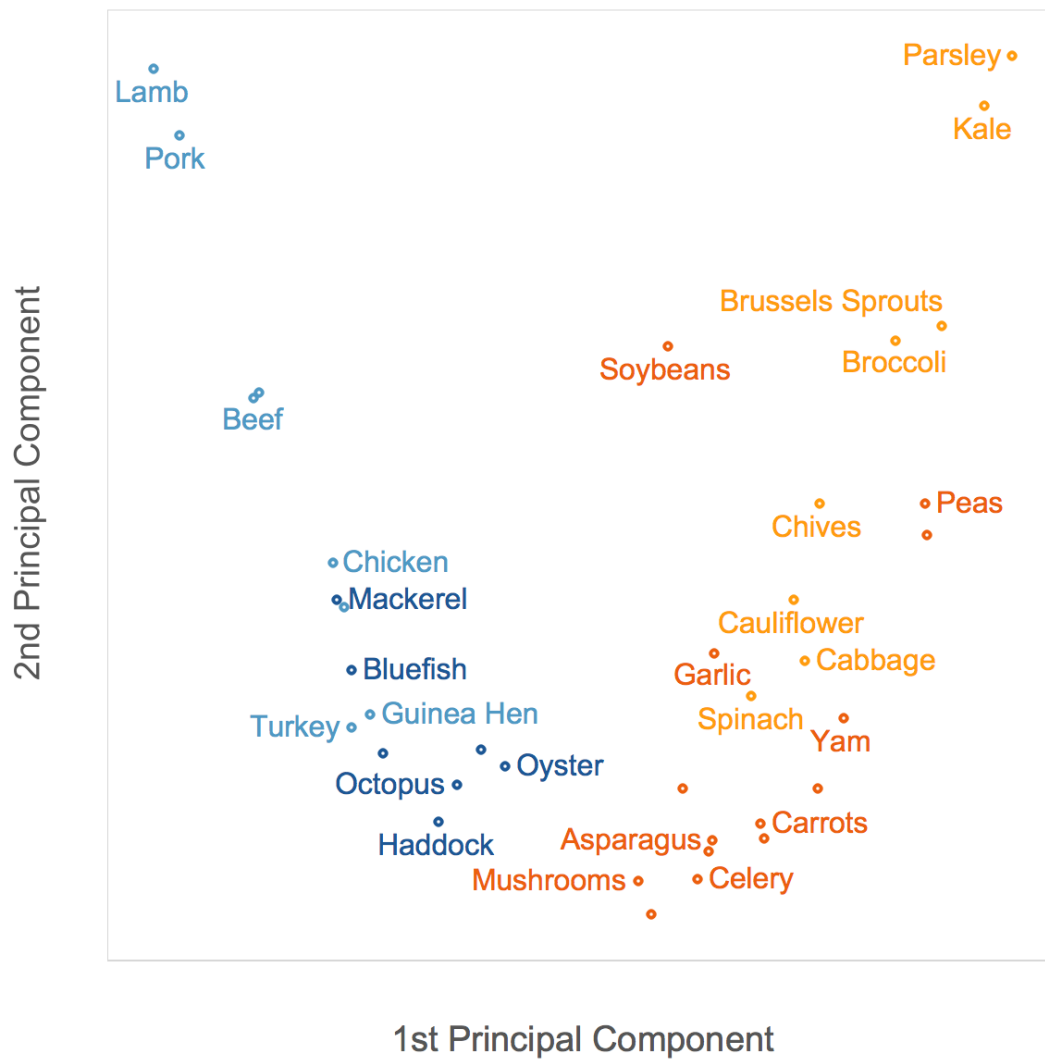


Figure 6. How principal components differentiate food items.

To keep results simple and generalizable however, only the first few principal components are selected for visualization and further analysis. Principal components are ordered by their effectiveness in differentiating data points, with the first principal component doing so to the largest degree. The number of principal components to shortlist is determined by a *scree plot* (see Figure 7).

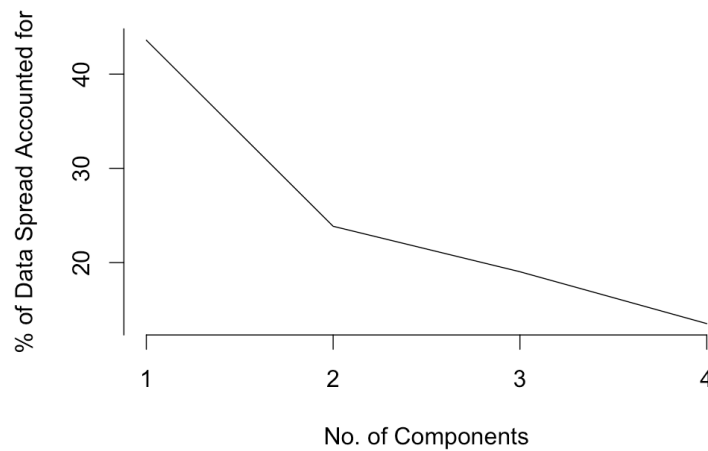


Figure 7. Scree plot showing a 'kink' where the number of components equals to 2.

A scree plot shows the decreasing effectiveness of subsequent principal components in differentiating data points. A rule of thumb is to use the number of principal components corresponding to the location of a *kink*. In the plot above, the kink is located at the second component. This means that even though having three or more principal components would better differentiate data points, this extra information may not justify the resulting complexity of the solution. As we can see from the scree plot, the top 2 principal components already account for about 70% of data spread. Using fewer principal components to explain the current data sample better ensures that the same components can be generalized to another data sample.

Limitations

Maximizing Spread. The main assumption of PCA is that dimensions which reveal the largest spread among data points are the most useful. However, this may not be true. A popular counter example is the task of counting pancakes arranged in a stack, with pancake mass representing data points.

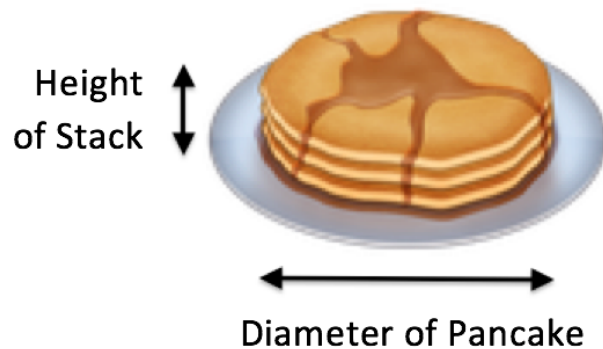


Figure 8. Pancake analogy.

To count the number of pancakes, one pancake is differentiated from the next along the vertical axis (i.e. height of the stack). However, if the stack is short, PCA would erroneously identify a horizontal axis (i.e. diameter of the pancakes) as a useful principal component for our task, as it would be the dimension along which there is largest spread.

Interpreting Components. Interpretations of generated components have to be inferred, and sometimes we may struggle to explain the combination of variables in a principal component.

Nonetheless, having prior domain knowledge could help. In our example with food items, prior knowledge of major food categories help us to comprehend why nutrition variables are combined the way they are to form principal components.

Orthogonal Components. One major drawback of PCA is that the principal components it generates must not overlap in space, otherwise known as orthogonal components. This means that the components are always positioned at 90 degrees to each other. However, this assumption is restrictive as informative dimensions may not necessarily be orthogonal to each other (see Figure 9).

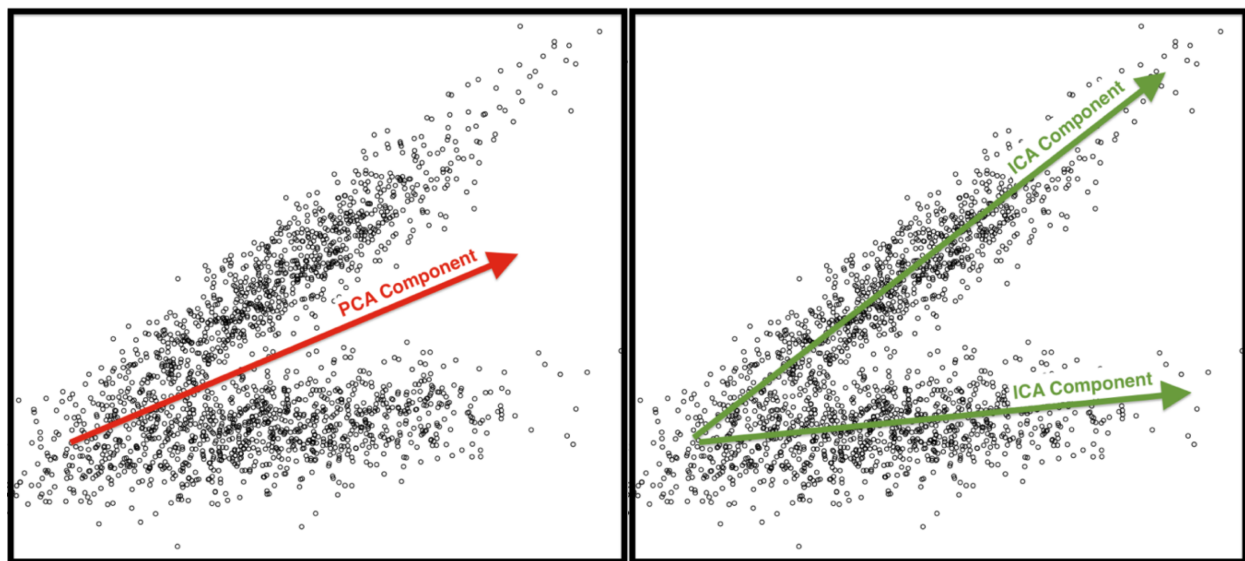


Figure 9. Comparing the effectiveness of PCA and ICA in identifying components.

To resolve this, we can use an alternative technique called *Independent Component Analysis* (ICA). ICA allows its components to overlap in space, thus they do not need to be orthogonal. Instead, ICA forbids its components to overlap in the information they contain, aiming to reduce mutual information shared between components. In other words, ICA's components are *independent*, with each component revealing unique information on the data set. Besides overcoming the orthogonality assumption, ICA also considers other information apart from data spread in determining principal components. Hence, it is less susceptible to the pancake error.

While ICA may seem superior to PCA, PCA remains one of the most popular technique for dimension reduction. Hence, knowing how PCA works and its assumptions is useful. When in doubt,

one could consider running a ICA to verify and complement results from a PCA.

Association Rules

Discovering Purchasing Patterns

In grocery shopping, each shopper has a distinctive list of things to buy depending on one's needs and preferences. A housewife might buy healthy ingredients for a family dinner, while a bachelor might buy beer and chips. Understanding these buying patterns can help to increase sales in several ways. If a pair of items, X and Y, are frequently bought together:

- X and Y could be placed on the same shelf, so buyers of one item would be prompted to buy the other
- Advertisements on X could be targeted at buyers who purchase Y
- X and Y could be combined into a new product, such as having X in flavors of Y



Figure 1. Product placement in Tesco, UK.

To uncover how items are associated with each other, we can use *association rules* analysis. Besides increasing sales profits, association rules can also be used in other fields. In medical diagnosis for instance, understanding comorbid symptoms can help to improve patient care and medicine prescription.

While we may know that certain items are frequently bought together, we need a systematic way to uncover these associations.

Support, Confidence and Lift

There are three common ways to measure association.

Measure 1: Support. This indicates *how popular an itemset is*, measured by the proportion of transactions where an itemset appears. In Table 1, {apple} appears in 4 out of 8 transactions, hence its support is 50%. Itemsets can also contain multiple items. For instance, the support of {apple, beer, rice} is 2 out of 8, or 25%.

$$\text{Support } \{\text{🍎}\} = \frac{4}{8}$$

Figure 2. Support measure.























Transaction 1	   
Transaction 2	  
Transaction 3	 
Transaction 4	 
Transaction 5	   
Transaction 6	  
Transaction 7	 
Transaction 8	 

Table 1. Example Transactions.

Support Threshold. A support threshold could be chosen to identify popular itemsets, such that itemsets with support values above this threshold would be deemed popular. If sales of items that constitute a large proportion of your transactions tend to have a significant impact on your profits, you might consider using that proportion as your support threshold.

Measure 2: Confidence. This indicates *how likely item Y is purchased when item X is purchased*, expressed as $\{X \rightarrow Y\}$. This is measured by the proportion of transactions with item X where item Y also appears. In Table 1, the confidence of $\{\text{apple} \rightarrow \text{beer}\}$ is 3 out of 4, or 75%.

$$\text{Confidence } \{\text{apple} \rightarrow \text{beer}\} = \frac{\text{Support } \{\text{apple}, \text{beer}\}}{\text{Support } \{\text{apple}\}}$$

Figure 3. Confidence measure.

One drawback of the confidence measure is that it might misrepresent the importance of an association. This is because it only accounts for how popular apples are, but not beers. If beers are also very popular in general, there will be a higher chance that a transaction containing apples will also contain beers, thus inflating the confidence measure. To account for the base popularity of both constituent items, we use a third measure called *lift*.

Measure 3: Lift. This indicates *how likely item Y is purchased when item X is purchased, while controlling for how popular item Y is*. For instance, the lift of $\{\text{apple} \rightarrow \text{beer}\}$ is equal to the confidence of $\{\text{apple} \rightarrow \text{beer}\}$ divided by the popularity of $\{\text{beer}\}$. Hence, the lift of $\{\text{apple} \rightarrow \text{beer}\}$ equals to 1, which implies no association between items. A lift value greater than 1 means that item Y is *likely* to be bought if item X is bought, while a value less than 1 means that item Y is *unlikely* to be bought if item X is bought.

$$\text{Lift } \{\text{apple} \rightarrow \text{beer}\} = \frac{\text{Support } \{\text{apple}, \text{beer}\}}{\text{Support } \{\text{apple}\} \times \text{Support } \{\text{beer}\}}$$

Figure 4. Lift measure.

Example: Mapping Grocery Transactions

Using data on actual transactions at a grocery outlet over 30 days obtained from the *arules* R library, a network graph was generated (Figure 5) which shows associations between selected items. Larger circles imply higher support, while red circles imply higher lift.

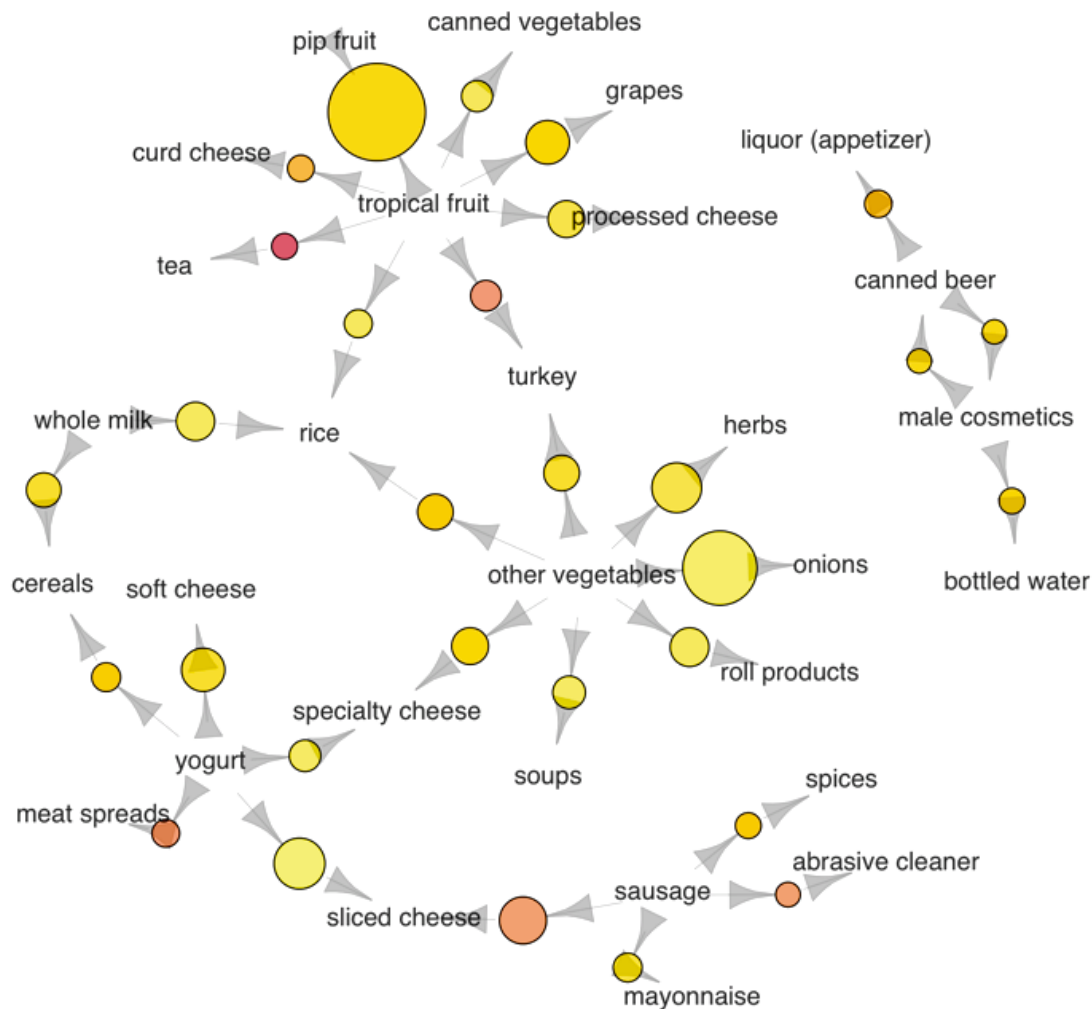


Figure 5. Associations between selected items. Visualized using the *arulesViz* R library.

Several purchase patterns can be observed. For example:

- The most popular transaction was of pip and tropical fruits
- Another popular transaction was of onions and other vegetables
- If someone bought meat spreads, he is likely to have bought yogurt as well
- Many people bought sausage along with sliced cheese

- If someone bought tea, he is likely to have bought fruit as well, possibly inspiring the production of fruit-flavored tea

Recall that one drawback of the confidence measure is that it could misrepresent the importance of an association. To demonstrate this, we picked 3 association rules from the original dataset which contained beer:

Transaction	Support	Confidence	Lift
Canned Beer → Soda	1%	20%	1.0
Canned Beer → Berries	0.1%	1%	0.3
Canned Beer → Male Cosmetics	0.1%	1%	2.6

Table 2. Association measures for beer-related rules.

The {beer → soda} rule had the highest confidence at 20%. However, both beer and soda appeared frequently across all transactions (see Table 3), so their association could simply be a fluke. This is confirmed by the lift value of {beer → soda}, which was 1, implying no association between beer and soda.

Transaction	Support
Canned Beer	10%
Soda	20%
Berries	3%
Male Cosmetics	0.5%

Table 3. Support of individual items.

On the other hand, the {beer → male cosmetics} rule had a low confidence, due to few purchases of male cosmetics in general. However, whenever someone did buy male cosmetics, he was very likely to have bought beer as well, as inferred from a high lift value of 2.6. The converse was true for {beer → berries}. With a lift value below 1, we might conclude that if someone bought berries, he would *not* be likely to have bought beer.

While it is easy to determine the popularity of individual itemsets, a business owner would typically be more interested in having a complete list of popular itemsets. To get this list, we need to calculate the support values for every possible configuration of items, and then shortlist the itemsets that have support values above a chosen threshold.

In a store with just 10 items, the total number of possible configurations to examine would be a whopping 1023. This number increases exponentially in a store with hundreds of items. Hence, we need a way to reduce the number of item configurations to consider.

Apriori Algorithm

The *apriori principle* is a solution to reduce the number of itemsets we need to examine. Put simply, the apriori principle states that if an itemset is infrequent, then all its subsets must also be infrequent. This means that if {beer} was found to be infrequent, we can expect {beer, pizza} to be equally or even more infrequent. So in consolidating the list of popular itemsets, we need not consider {beer, pizza}, nor any other itemset configuration that contains beer.

Finding itemsets with high support.

Using the apriori principle, the number of itemsets that has to be examined can be pruned, and the list of popular itemsets can be obtained in these steps:

Step 0. Start with itemsets containing just a single item, such as {apple} and {pear}.

Step 1. Determine the support for itemsets. Keep the itemsets that meet a minimum support threshold, and remove itemsets that do not.

Step 2. Using the itemsets you have kept from Step 1, generate all possible itemset configurations.

Step 3. Repeat Steps 1 & 2 until there are no more new itemsets.

Figure 6 shows how the number of possible combinations from 4 items could be significantly pruned using the apriori principle. When {apple} had low support, it was removed along with all other itemset configurations that contained apple. This reduced the number of itemsets to consider by more than half.

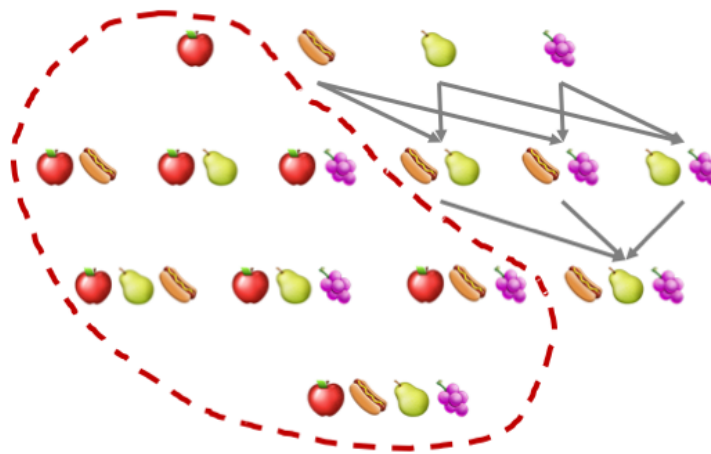


Figure 6. Itemsets within the red border would have been pruned.

Finding item rules with high confidence or lift.

We have seen how the apriori algorithm can be used to identify itemsets with high support. The same principle can also be used to identify item associations with high confidence or lift. Finding rules with high confidence or lift is less computationally taxing once high-support itemsets have been identified, because confidence and lift values are calculated using support values.

Take for example the task of finding high-confidence rules. If the rule {beer, chips \rightarrow apple} has low confidence, all other rules with the same constituent items and with apple on the right-hand side would have low confidence too. Specifically, the rules {beer \rightarrow apple, chips} and {chips \rightarrow apple, beer} would have low confidence as well. As before, lower level candidate item rules can be pruned using the apriori algorithm, so that fewer candidate rules need to be examined.

Limitations

Computationally Expensive. Even though the apriori algorithm reduces the number of candidate itemsets to consider, this number could still be significant when store inventories are large or when the support threshold is low. An alternative solution would be to reduce the number of comparisons by using advanced data structures to sort candidate itemsets more efficiently.

Spurious Associations. When analyzing a large number of itemsets, associations could happen by chance. To ensure that associations found are generalizable, they should be [validated](#).

Despite these limitations, association rules remain an intuitive method to identify important patterns, which works well for datasets of manageable size.

Correlation and Regression

Selecting People to Recruit

Military defense is critical for a country's survival, and for a military to be strong, it needs to be led by competent commanders. Commander training is intensive and time-consuming. Hence, it is important to be accurate in identifying eligible candidates for training.

The question is, what are the traits of a potential commander? One might surmise that a commander's performance could be gauged by several factors, such as physical fitness and cognitive ability. However, we do not know for sure:

- How these factors are related to performance
- How these factors rank against each other in predicting performance
- How these factors could be used collectively to predict performance

To answer these, we can use *regression analysis*, a prediction technique. In selecting commanders, we use plausible predictors, such as fitness test scores and IQ scores. The prediction model would look something like this:

$$\text{commander potential} = \text{fitness score} + \text{IQ score}$$

Regression analysis would then allow us to find out whether fitness and IQ are indeed strong predictors of commander potential and, if so, which predictor is stronger.

Example: Profiling Military Commanders

Military recruits undergo various assessments, and their scores on these assessments could be used to gauge their final GPA in commander training school, a proxy for commander potential.

We can visualize how each individual assessment relates to eventual commander potential with the use of scatterplots. Examples of these plots are illustrated below using mock data.

Figure 1 shows soldiers' commander potential against their fitness scores. We observe a consistent trend - fitter soldiers are more likely to be better commanders.

The red line running through the data points is what is known as a *best-fit line*, which is essentially an overall trend line. Predictions are made using this trend line. For example, if a soldier scored 85 for fitness, the trend line predicts that he would score 70 for commander potential.

Similar to fitness, IQ is also positively correlated with commander potential (see Figure 2). In fact, IQ appears to be a stronger predictor of commander potential than fitness. This can be seen from how data points are more concentrated along the trend line. The trend line itself is also steeper, implying that a small increase in IQ corresponds to a proportionally bigger increase in commander performance.

If both fitness and IQ scores can predict commander potential, combining both scores could improve their predictive power.

However, simply summing up both scores would not be ideal. From the above plots, we know that IQ is a stronger predictor of commander potential than fitness, and should thus be given more weight. But how much more?

To find the optimal set of weights, we can use regression analysis. Weights account for the differing strengths of predictors to improve predictions. The improved prediction model would look something like this:

$$\text{commander potential} = w1(\text{fitness score}) + w2(\text{IQ score}),$$

where the weight for IQ ($w2$) is larger than the weight for fitness ($w1$).

From Figure 3, one can observe that the data points are even closer to the trend line than before, implying that predictions using the trend line is closely matched with actual results.

Deriving a Trend Line

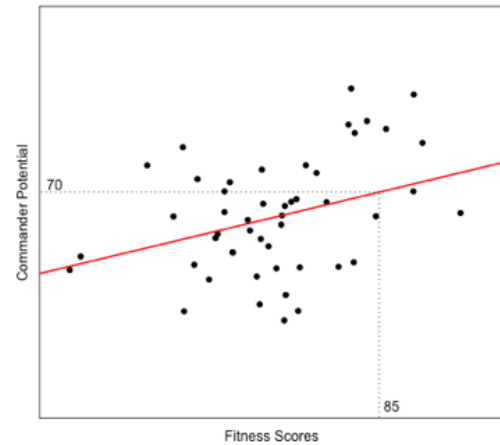


Figure 1. Commander potential against fitness scores.

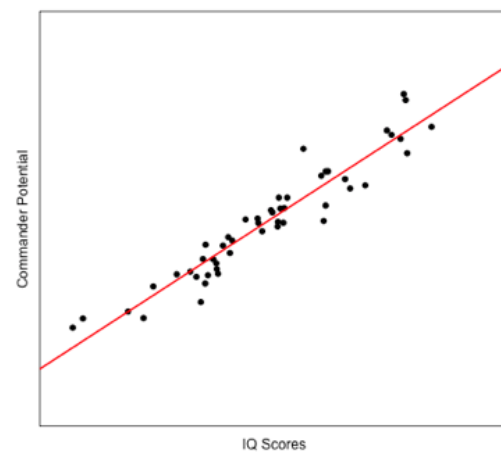


Figure 2. Commander potential against IQ scores.

After seeing how trends lines from regression analysis can be used to make predictions, we now examine how trend lines are derived.

A common solution is to draw the trend line such that the total squared prediction error is minimized. This is called the method of least-squares, as illustrated in Figure 4.

The trend line represents predicted scores, while data points indicate actual scores. So the dotted lines, which show the distance between predicted and actual scores, represent prediction errors. Each prediction error is first squared, then all the squared values are summed. This resulting sum is minimized by the optimal trend line.

Where there are two or more predictors, weights are assigned to every predictor to account for their differing predictive strengths. The overall trend line would then be a weighted sum of the trend lines for individual predictors. So if using IQ scores lead to lower prediction error, the final trend line would lean more towards predictions made from IQ scores than from fitness scores (see Figure 5).

When there is only one predictor, its weight is a measure of how good a proxy that predictor is for the variable we are trying to predict. A weight of zero implies that the predictor is not related at all to the target variable.

When there are multiple predictors, one may be tempted to rank the strength of predictors using their weights. While valid in some cases, this approach has several pitfalls:

It is conditional on the other predictors present.

A predictor's weight is a measure of how strong that predictor is, in the presence of other predictors. In other words, it is the "value-added" by that predictor, rather than its absolute predictive strength. For example, if high school grades are included along with IQ scores as predictors, the weight for high school grades may be negligible, even though it may be a strong predictor of commander potential on its own. This is because high school grades overlaps with IQ scores in measuring cognitive ability, thus adding little value to overall predictive power.

Results vary with different units of measure. If represented in different units, two identical predictors making the same prediction would be assigned different weights. For example, a predictor measured in centimetres will always have a weight that is 100 times smaller than the same one measured in metres.

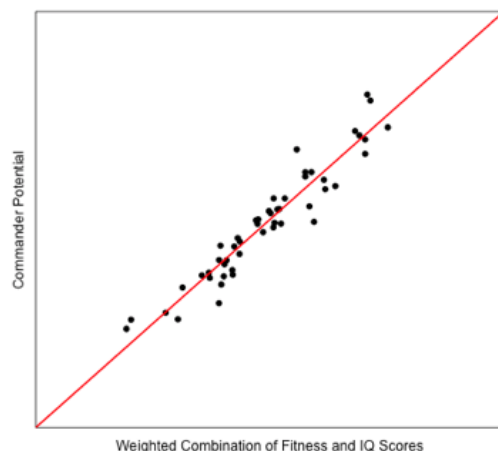


Figure 3. Commander potential against fitness and IQ scores.

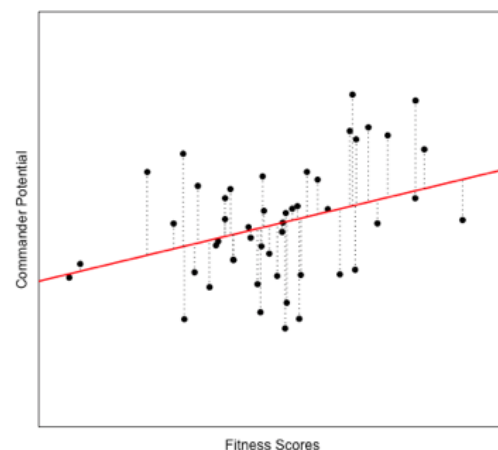


Figure 4. Minimizing deviations from the trend line.

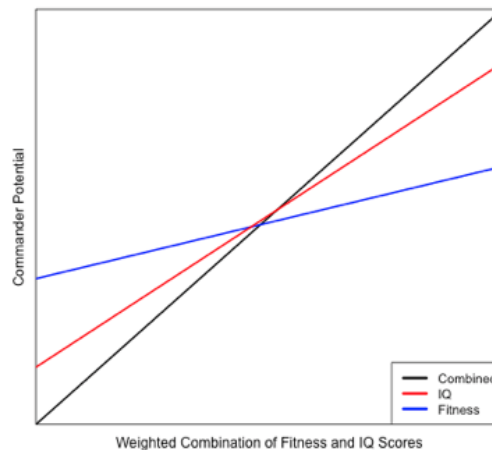


Figure 5. Comparing prediction accuracy across trend lines.

There are ways to avoid the above pitfalls. First, one can *standardize* the units of predictor variables, in a way that is analogous to expressing each variable in terms of percentiles. Converting predictors to the same unit allows for more accurate comparisons. The weights of standardized predictors are then called *beta weights*.

When there is only one predictor, the beta weight of that predictor is also known as a *correlation coefficient*, denoted as r . Correlation coefficients range from -1 to 1, and this provides two bits of information about the correlation:

- **Direction.** Positive (negative) coefficients imply that correlated variables move in the same (opposite) direction; as one increases, the other increases (decreases).
- **Magnitude.** The magnitude of a correlation can be inferred from the absolute value of the coefficient. The closer the coefficient is to -1 or 1, the stronger the predictor. Since correlation coefficients indicate the absolute strength of individual predictors, they are a more reliable way to rank predictors than regression weights.

To summarize, regression analysis is a technique for making predictions using one or more predictors. It derives a trend line based on a general principle of reducing prediction errors. When there are multiple predictors, predictors are assigned weights according to their relative predictive strengths. When there is only one predictor, its correlation coefficient reveals the direction and strength of relationship with the target variable. Thus, correlation coefficients can also be used to rank predictors.

Limitations

After learning about what regression analysis can be used for and how it works, we now examine its limitations:

Sensitivity to outliers. As the regression analysis accounts for all data points equally, a single data point with extreme values could skew the trend line significantly. Before running the analysis, outliers should be identified using scatterplots.

Distorted weights of correlated predictors. As alluded to above, the inclusion of highly-correlated predictors in a regression model would distort the interpretation of their weights. This problem is called multicollinearity. To ensure that predictions remain generalizable to new cases, regression models should be kept simple with as few uncorrelated predictors as needed. Alternatively, more advanced techniques such as lasso or ridge regression could be used to overcome multicollinearity.

Different combinations of predictors. The type of regression explained in this post is called simple linear regression. The term 'linear' means that effects of predictors are summed up, resulting in a trend line that is straight. However, some trends may be curved. For instance, having an average Body Mass Index (BMI) is a mark of a fit commander. BMI values that are too-low or high are undesirable. To test non-linear trends however, more complex models have to be used.

Finally, you might have heard of the phrase "*correlation does not imply causation*". To illustrate this, suppose that household income was found to be positively correlated with commander potential. This does not mean that having your spouse switch to a higher-paying job would make you a better commander. Rather, higher household income could pay for better education opportunities, which in turn improves cognitive skills required by commanders. Being mindful of how results may be misinterpreted helps to ensure the accuracy of conclusions.

k -Nearest Neighbors and Anomaly Detection

Identifying Categories by Chemical Make-Up

Have you ever wondered about the difference between red and white wine?

Some assume that red wine is made from red grapes, and white wine is made from white grapes. But this is not entirely true, because white wine can also be made from red grapes, though red wine cannot be made from white grapes.

The main difference between red and white wine is in how the grapes are fermented. To make red wine, grape juice is fermented along with the grape skin, the source of red pigments. For white wine, the juice is fermented without the skin. Presence of grape skin in the fermentation process is likely to affect not only the color of the wine, but also its chemical makeup. This means that, without looking at the color of the wine, we can possibly infer whether a wine is red or white, based on levels of its chemical compounds.

To check our hypothesis, we can use a technique called *k-Nearest Neighbors* (k -NN). k -NN is one of the simplest methods in machine learning. It classifies a data point based on how its neighbors are classified. So if we are guessing the color of a particular wine, we can refer to the color of other wines that have the most similar chemical makeup (i.e. neighbors). Besides classification into groups, k -NN can also be used to estimate continuous values. To approximate a data point's value, k -NN takes the aggregated value of its most similar neighbors.

Example: Distilling Wine Differences

Data on the chemical makeup of red and white variants of Portuguese *Vinho Verde* wine was obtained from Cortez et al., 2009. We can plot the 1599 red and 4898 white wines with 2 chemical compounds, chlorides and sulfur dioxide, as the axes (see Figure 1).

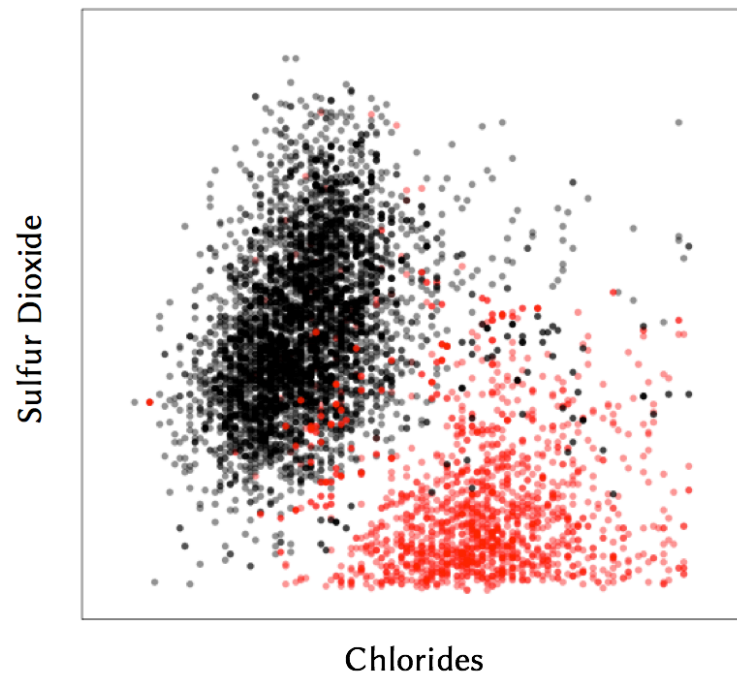


Figure 1. Levels of chlorides and sulfur dioxide in red wines (red) and white wines (black).

Minerals such as sodium chloride (i.e. table salt) are concentrated in grape skin, hence more of these get infused into red wines. Grape skin also contains natural anti-oxidants that keep the fruit fresh. Without it, white wines require more sulfur dioxide, which serves as a preservative. For these reasons, red wines are clustered at the bottom-right of the plot, while white wines are at the top-left.

To deduce the color of a wine with specified levels of chloride and sulfur dioxide, we can refer to the known colors of neighboring wines with similar quantities of both chemical compounds. By doing this for each point in the plot, we can draw decision boundaries distinguishing red from white wines (see Figure 2).

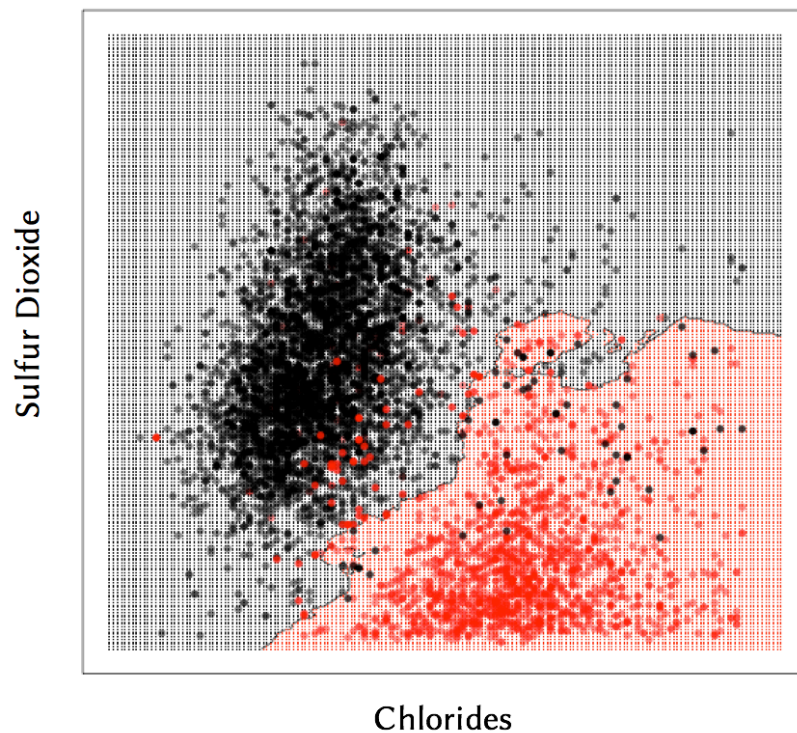


Figure 2. Predicted classifications of wine color using k -NN. Unknown wines inside the red boundary would be predicted as red wine, whereas those in the black boundary would be predicted as white wine.

Using these decision boundaries, we can predict a wine's color with over 98% accuracy. Let us take a look at how these boundaries are inferred from nearest neighbors.

Classification Based on Neighbors

Recall that k -NN classifies a data point based on how its neighbors are classified. This means if a data point is surrounded by 4 red points and 1 black point, that data point is likely a red point by majority vote.

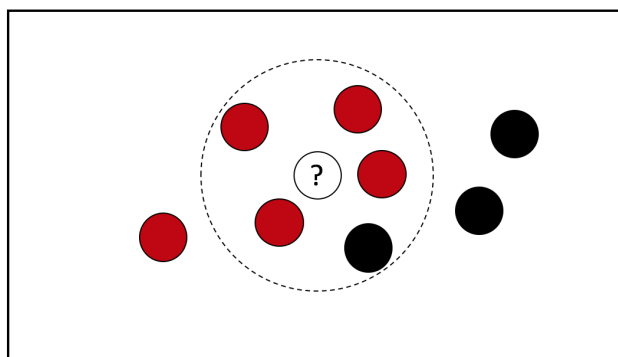


Figure 3. A data point is classified by majority votes from its 5 nearest neighbors. Here, the unknown point would be classified as red, since 4 out of 5 neighbours are red.

The k in k -NN is a parameter that refers to the number of nearest neighbors to include in the majority voting process. In the above example, k equals to 5. Choosing the right value of k is a process called parameter tuning, and is critical to prediction accuracy.

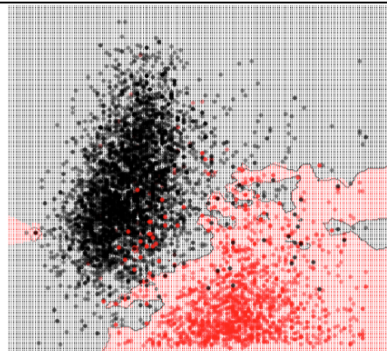
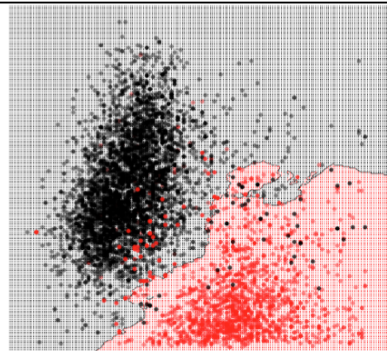
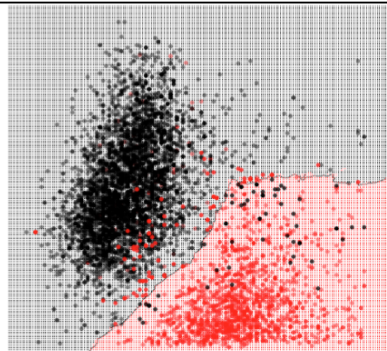
$k = 3$	$k = 17$	$k = 50$
		
98.2% accuracy	98.6% accuracy	97.8% accuracy
Overfit	Ideal fit	Underfit

Table 1. Comparison of model fit using varying values of k .

If k is too small, data points would match immediate neighbors only, amplifying errors due to random noise. If k is too large, data points would try to match far flung neighbors, diluting underlying patterns. But when k is just right, data points would reference a suitable number of neighbors such that errors cancel out to reveal subtle trends in the data.

To achieve the best fit and lowest error, the parameter k can be tuned by [cross-validation](#). In our binary (i.e. two class) classification problem, we can also avoid tied votes by choosing k to be an odd number.

Apart from classifying data points into groups, k -NN can also be used to predict continuous values, by aggregating the values of the nearest neighbors. Instead of treating all neighbors equally and taking a simple average, the estimate could be improved by taking a weighted average, where we pay more attention to the values of closer neighbors than those further away, since closer neighbors

are more likely to reflect a data point's true value.

Anomaly Detection

k-NN is not limited to merely predicting groups or values of data points. It can also be used in detecting anomalies. Identifying anomalies can be the end goal in itself, such as in fraud detection. Anomalies can also lead you to additional insights, such as discovering a predictor you previously overlooked.

The simplest approach to detecting anomalies is by visualizing the data in a plot. In Figure 2 for instance, we can see immediately which wines deviate from their clusters. However, simple 2-D visualizations may not be practical, especially when you have more than 2 predictor variables to examine. This is where predictive models such as *k*-NN come in.

As *k*-NN uses underlying patterns in the data to make predictions, any errors in these predictions are thus telltale signs of data points which do not conform to overall trends. In fact, by this approach, any algorithm that generates a predictive model can be used to detect anomalies. For instance, in [regression analysis](#), an outlier would deviate significantly from the best-fit line.

In our wine data, we can examine misclassifications generated from *k*-NN analysis to identify anomalies. For instance, it seems that red wines that get misclassified as white wines tend to have higher-than-usual sulfur dioxide content. One reason could be the acidity of the wine. Wines with lower acidity require more preservatives. Learning from this, we might consider accounting for acidity to improve predictions.

While anomalies could be caused by missing predictors, they could also arise due to insufficient data for training the predictive model. The fewer data points we have, the more difficult it would be to discern patterns in the data. Hence, as it is important to ensure an adequate sample size.

Once anomalies have been identified, they can be removed from the datasets used to train predictive models. This will reduce noise in the data, thus strengthening the accuracy of predictive models.

Limitations

Although *k*-NN is simple and effective, there are times when it might not work as well:

Imbalanced classes. If there are multiple classes to be predicted, and the classes differ drastically in size, data points belonging to the smallest class might be overshadowed by those from bigger classes, increasing their risk of misclassification. To improve accuracy, we could use swap majority voting in favor of weighted voting, whereby the class of closer neighbors are weighted more heavily than ones further away.

Excess predictors. If there are too many predictors to consider, it would be computationally intensive to identify and process nearest neighbors across multiple dimensions. Moreover, some predictors could be redundant as they do not improve prediction accuracy. To resolve this, we can use [dimension reduction techniques](#) to extract only the most powerful predictors for analysis.

References

Lichman, M. (2013). *UCI Machine Learning Repository*. Irvine, CA: University of California, School of Information and Computer Science.

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In *Decision Support Systems*, Elsevier, 47(4):547-553. ISSN: 0167-9236.

Support Vector Machine

“No” or “Oh no”?

Medical diagnosis is complex. It requires multiple factors to be accounted for, and it can be vulnerable to the subjective opinions of doctors. Sometimes, the correct diagnosis is not made until it is too late. A more systematic approach to diagnosing ailments might be to use prediction techniques in data science, in which whole databases of medical symptoms are systematically combed to derive prediction models for underlying medical conditions.

In this chapter, we will examine one prediction technique called *support vector machine* (SVM). It seeks to derive an optimal boundary to separate data points into two groups (e.g. healthy vs. unhealthy).

Example: Predicting Heart Disease

One of the most common health conditions in developed countries is heart disease, in which heart vessels are narrowed or blocked, leading to increased risk of heart attack. Heart disease can be diagnosed conclusively with an imaging scan, but the costs of these scans prohibit most people from doing it regularly. An alternative would be to shortlist high-risk patients to undergo the scan based on physiological symptoms.

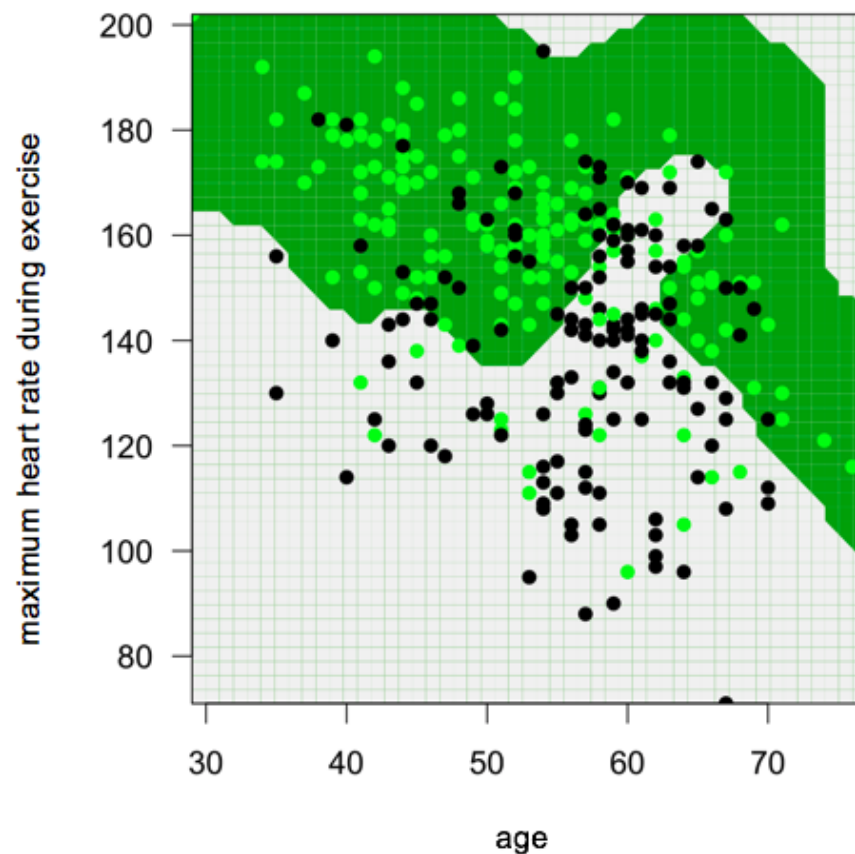


Figure 1. Using SVM to predict the presence of heart disease by deriving patient profiles based on maximum heart rate during exercise and age. The dark green region represents the predicted profile of healthy adults, while the gray region represents the predicted profile of heart disease patients. The light green and black points represent actual data from healthy adults and heart disease patients respectively.

To determine which symptoms predict the presence of heart disease, patients from an American clinic were made to exercise while their physiological states were recorded. Subsequently, they underwent imaging scans to determine the presence of heart disease. One of the physiological indicators measured was the patient's maximum heart rate attained during exercise. Using this heart rate data together with a patient's age, an SVM prediction model is able to tell with over 75% accuracy if someone is suffering from heart disease.

Heart disease patients generally had lower heart rates during exercise compared to others of the same age, and the disease seemed more prevalent among patients above 55 years old.

While heart rate tended to decrease with age, heart disease patients aged about 60 years old appeared to have similar heart rates with younger healthy adults, as indicated by the abrupt arc in the decision boundary. If not for SVM's ability to pick up curved patterns, this phenomenon might have been overlooked.

Delineating an Optimal Boundary

The main objective of SVM is to derive an **optimal boundary** that separates one group from the other. This is not as straightforward as it sounds, as there are multiple possible boundaries (see Figure 2).

To find the optimal boundary, we have to first identify the peripheral data points which are located closest to points from the other group. The optimal boundary is drawn down the middle between peripheral data points of both groups (see Figure 3).

As these peripheral data points support the discovery of the optimal boundary, these are called *support vectors*.

One advantage of using SVM is computational speed. As its decision boundary is determined only by peripheral data points, it takes less time to derive. This is in contrast with techniques like [regression](#), which accounts for every data point to derive a trend line.

However, relying on just a subset of data points could also be a pitfall, as the decision boundary would be more sensitive to the positions of support vectors, which could fluctuate depending on which data points happen to be sampled as training data. Furthermore, it is rare to have data points cleanly separated into groups as in Figures 2 and 3. In reality, data points from both groups are likely to overlap, as seen in Figure 1.

To overcome these problems, the SVM algorithm has a key feature: a buffer zone that allows a limited number of training data points to cross over to the incorrect side. This results in a “softer” boundary that is robust against outliers in classifying training data, and thus more generalizable to new data.

The buffer zone is created by tuning a *cost parameter* that determines the degree of misclassification errors that would be tolerated. A larger cost parameter increases the tolerance level, resulting in a wider buffer zone. To ensure that the resulting model yields accurate predictions for current as well as new data, we can find the best value for the cost parameter via [cross-validation](#).

Another strength of SVM is its ability to account for curved patterns in the data. While there are other techniques that do so, the formula to derive a curved pattern

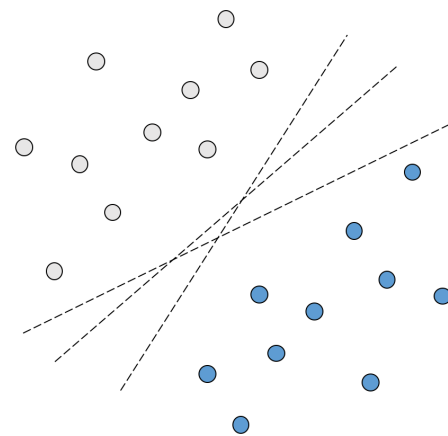


Figure 2. Multiple ways to separate two groups.

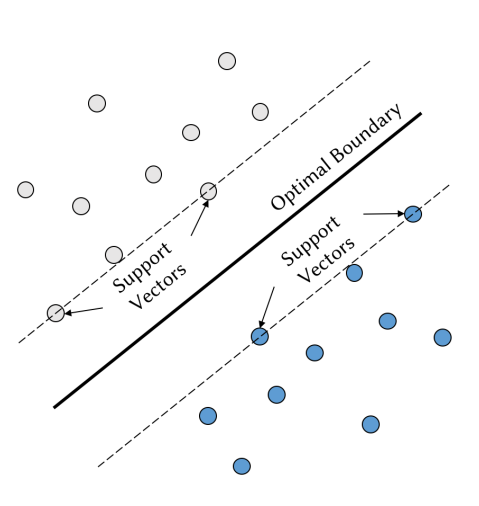


Figure 3. Optimal boundary is located in the middle of peripheral data points from opposing groups.

is inherently complex. SVM is favored for its superior computational efficiency, as it uses a method called the *kernel trick*.

Instead of drawing the curved boundary directly onto the data plane, SVM first projects the data onto a higher dimension, where the data points can be separated with a simple straight line (see Figure 4). Straight lines are easier to compute, and when projected back down onto a lower dimension, they could be translated into curved lines.

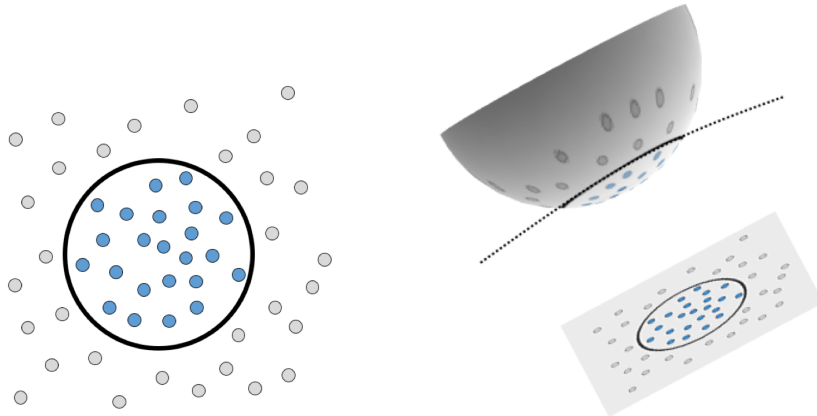


Figure 4. A circle of blue points on a 2-D sheet could be delineated by a straight line when projected onto a 3-D sphere.

Ease of manipulating data in higher dimensions is one reason why SVM is a popular tool for analyzing datasets with many variables. Common applications of SVM include decoding genetic information and evaluating sentiments in text.

Limitations

Although SVM is a versatile and fast prediction tool, it might not work well in certain scenarios:

Small datasets. As SVM relies on support vectors to determine decision boundaries, a small sample size would mean fewer of such support vectors to accurately position the boundaries.

Multiple groups. SVM is only able to classify 2 groups at a time. Where there are more than two groups, SVM could be iterated to distinguish each group from the rest using a technique called multi-class SVM.

Large overlap between groups. SVM classifies a data point based on which side of the decision boundary it falls on. However, when there is a large overlap between data points from both groups, a data point near the boundary might be more prone to misclassification. Despite this, SVM does not give additional information on each data point's probability of misclassification. One solution might be to use a data point's distance from the decision boundary to gauge its classification accuracy.

References

Lichman, M. (2013). [UCI Machine Learning Repository](#). Irvine, CA: University of California, School of Information and Computer Science.

Credits to Robert Detrano, M.D., Ph.D. from V.A. Medical Center, Long Beach and Cleveland Clinic Foundation for the data on heart disease.

Decision Tree

Predicting Survival in a Disaster

During a disaster, certain groups of people, such as women and children, might be entitled to receiving help first, granting them a higher chance of survival. One way to identify these groups is to use *decision trees*.

A decision tree predicts your chance of survival by asking a series of questions (see Figure 1). Each question must only have 2 possible responses, such as “yes” versus “no”. You start at the top question, called the root node, and move through the tree branches guided by your responses, until you reach a leaf node. The proportion of survivors at that leaf node would be your predicted chance of survival.

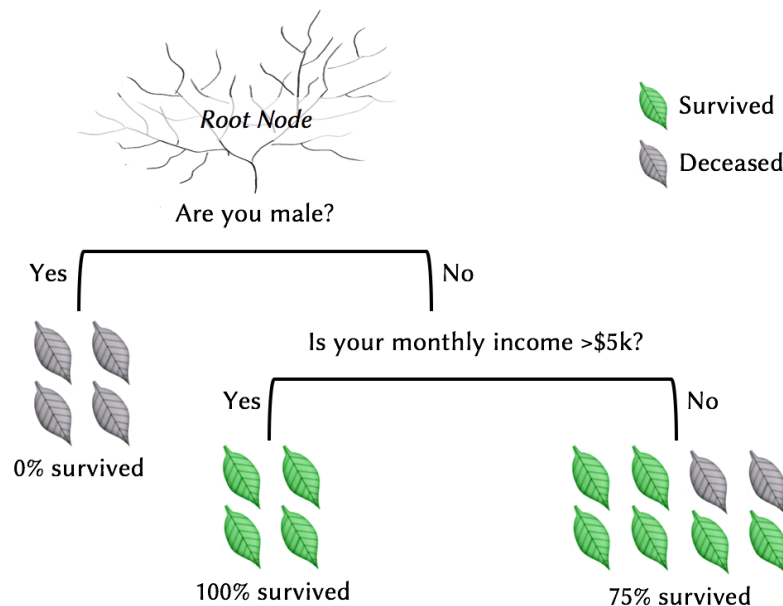


Figure 1. Example decision tree.

Decision trees have broader applications such as predicting survival rates for medical diagnosis, identifying who would resign or detecting fraudulent transactions.

Decision trees are versatile. They can handle questions about categorical groupings (e.g. male vs. female) or about continuous values (e.g. income). If the question asks about a continuous value, the values can be split into groups – for instance, comparing values which are “*above average*” versus “*below average*”.

In standard decision trees, there should only be two possible responses, such as “yes” versus “no”. If we want to test three or more responses (e.g. “yes”, “no”, “sometimes”), we can simply add more branches down the tree (see Figure 2).

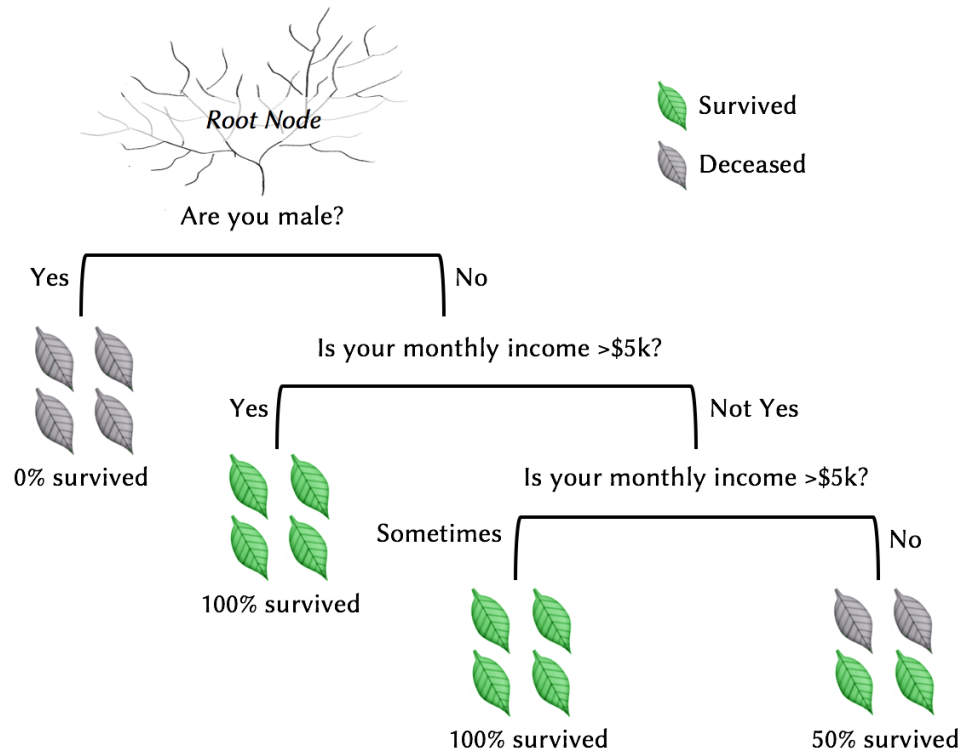


Figure 2. Testing multiple categories in a decision tree.

Example: Surviving the Titanic

We use passenger data for the ill-fated cruise liner, the *Titanic*, to check if certain groups of passengers were more likely to have survived. The dataset was originally compiled by the *British Board of Trade* to investigate the ship’s sinking. The data used in this example is a subset of the original, and is one of the in-built datasets freely available in R.

The computed decision tree predicting survival rates is illustrated in Figure 3.

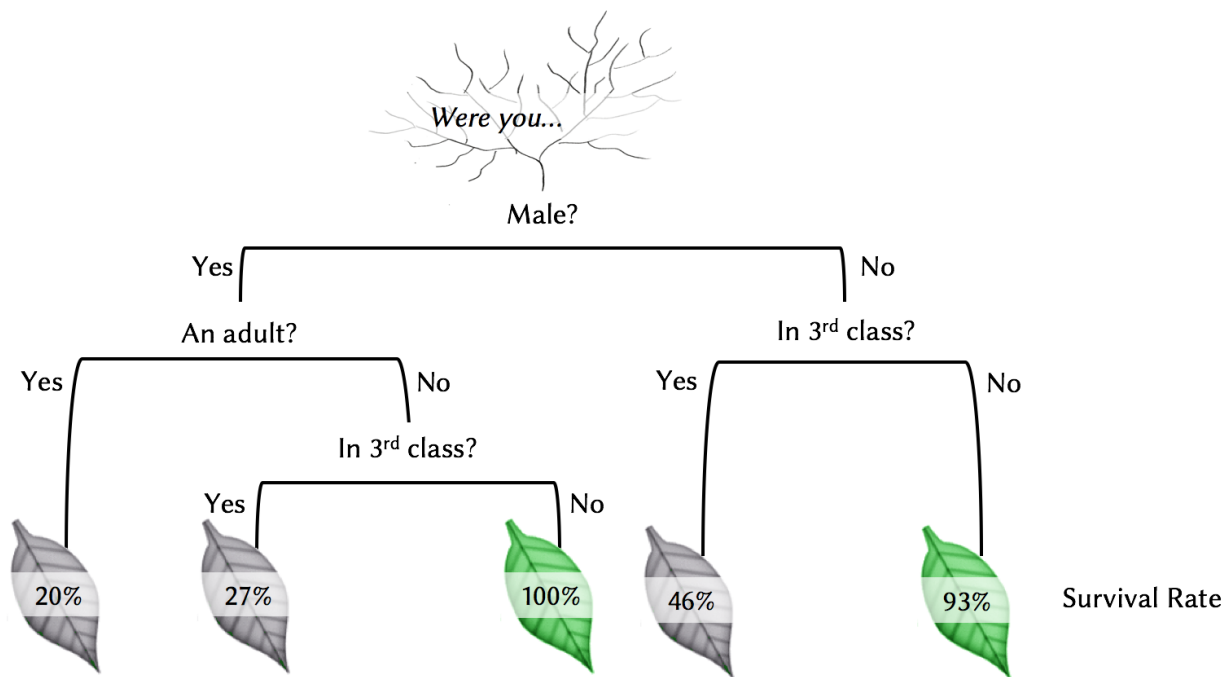


Figure 3. Predicting whether one would survive the sinking of Titanic.

From the result, it seems that you would have a good chance of being rescued from the *Titanic* if you were not from a 3rd class cabin, and you were either a male child or a female.

Decision trees are popular because they are easy to interpret. The question is, how is a decision tree generated?

Generating a Decision Tree

A decision tree is grown by first splitting all data points into two groups, with similar data points grouped together, and then repeating the binary splitting process within each group. As a result, each subsequent leaf node will have fewer but more homogeneous data points. The basis of decision trees is that by isolating groups of “survivors” via different paths in the tree, anyone else belonging to those paths would be predicted to be a likely “survivor” as well.

The process of repeatedly partitioning the data to obtain homogeneous groups is called recursive partitioning. It involves just 2 steps:

Step 1: Identify the binary question that splits data points into two groups that are most homogeneous.

Step 2: Repeat Step 1 for each leaf node, until a stopping criterion is reached.

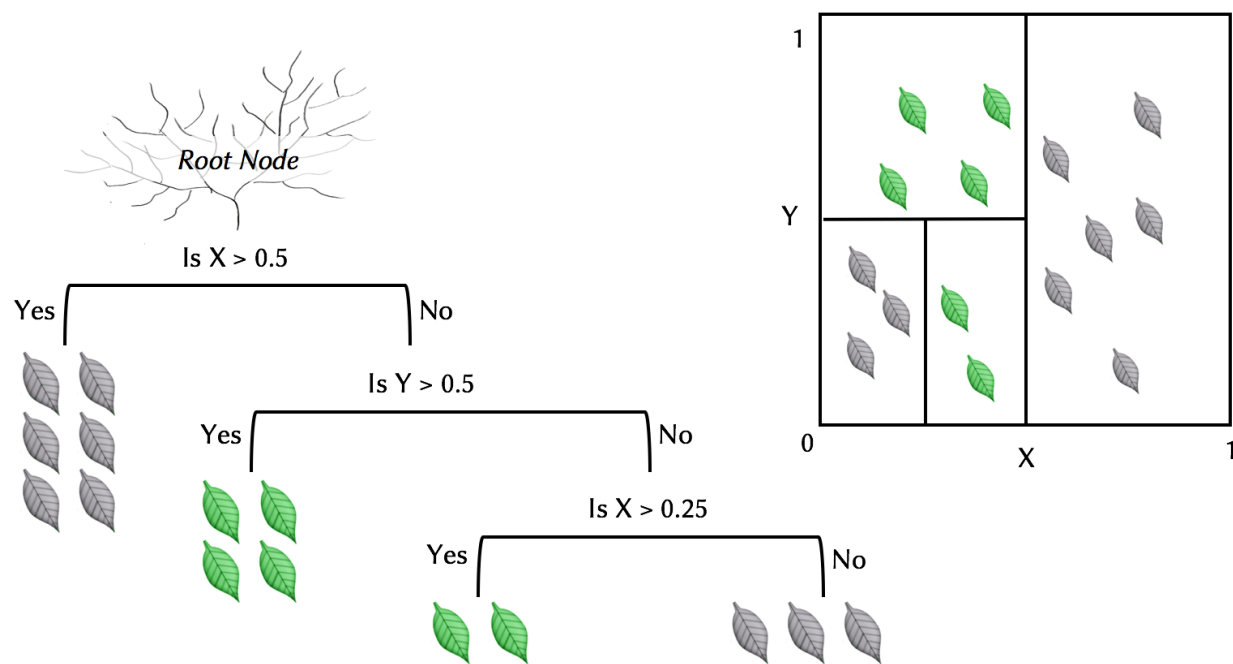


Figure 4. Partitioning of data points by a decision tree shown in a scatterplot.

There are various possible stopping criteria:

- Stop when data points at the leaf are all of the same predicted category/value
- Stop when the leaf contains less than five data points
- Stop when further branching does not improve homogeneity beyond a minimum threshold

Stopping criteria are selected using [cross-validation](#) to ensure that the decision tree can draw accurate predictions for new data.

As recursive partitioning only uses the best binary questions to grow a decision tree, the presence of non-significant variables would not affect results. Moreover, binary questions tend to divide data points around central values, so decision trees are robust against extreme values (i.e. outliers).

Limitations

While decision trees are easy to interpret, they have some drawbacks.

Instability. Decision trees are grown by splitting data points into homogeneous groups. However, a slight change in the data could result in a different tree dictating how data points should be split. By aiming for the best way to split data points each time, decision trees are prone to [overfitting](#).

Inaccuracy. Using the best binary question to split the data at the start may not lead to the most accurate predictions. Sometimes, less effective splits used initially may lead to better predictions subsequently.

To overcome these limitations, we can avoid aiming for the best split each time. Instead, we can diversify the trees grown. By combining predictions from different trees, we can get more stable and accurate results.

There are two methods to diversify trees

- The first method chooses different combinations of binary questions to grow multiple trees, and then aggregates the predictions from those trees. This technique is called a *random forest*.
- Instead of choosing binary questions at random, the second method strategically selects binary questions such that the prediction accuracy for each subsequent tree improves incrementally. Then, a weighted average of predictions from all trees is taken. This technique is called *gradient boosting*.

While random forest and gradient boosting tend to produce more accurate predictions, their complexity renders the solution harder to visualize. Hence, they are often called “*black-boxes*”.

On the other hand, decision trees can be visualized easily, allowing us to identify predictors and their interactions. Interpreting results is important for planning targeted interventions, which is why decision trees are still a popular tool for analysis.

Random Forest

Wisdom of the Crowd

Can several wrongs make a right?

While counter-intuitive, this is possible, even expected, in some of the best predictive models.

This is because each individual model has its own strengths and weakness. As there is only one correct prediction but many possible wrong predictions, individual models that yield correct predictions tend to reinforce each other, while wrong predictions cancel each other out. Hence, combining individual models is one way to improve prediction accuracy. This process is called *ensembling*.

One example is a random forest, which is an ensemble of [decision trees](#). To show how random forest is superior to its constituent trees, we generated 1000 possible decision trees to predict crime in a US city, and then we compared these predictions to that from a random forest grown from the same 1000 trees.

Example: Forecasting Crime

Using open data from the San Francisco Police Department, we got information on the location, date and severity of crimes that occurred in the city from 2014 to 2016. In addition, we obtained the city's weather records over the same period, as research has shown that crimes tend to occur on hotter days. Daily temperatures and precipitation levels were taken from the National Oceanic and Atmospheric Administration database.

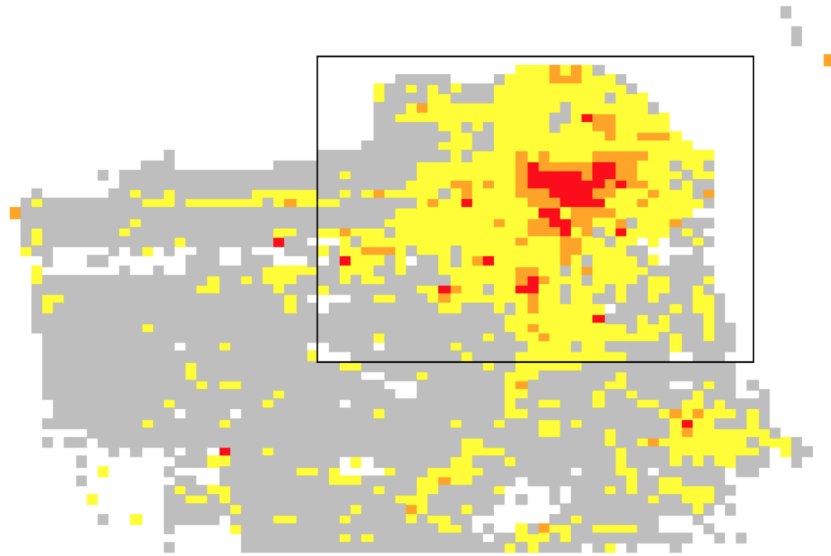


Figure 1. Heat map of San Francisco, California, USA, showing frequency of crimes: very low (gray), low (yellow), moderate (orange), or high (red).

From preliminary analysis, we can see how crime occurred mainly in the boxed area north-east of the city (Figure 1), so we further examined this area by dividing it into smaller regions measuring 900ft by 700ft (260m by 220m).

To predict where and when a crime might occur, 1000 possible decision trees were generated based on historical crime and weather data, and these trees were then combined in a random forest. Data from 2014 to 2015 was used to train the models, while data from 2016 (January to August) was used to test the models' accuracy.

It would not be feasible for a lean police force to implement extra security patrols for all areas predicted to have crime. Hence, we programmed our prediction model to identify only the top 30% of regions with the highest probability of violent crime occurring each day, so that the police could focus on enhancing patrols in these areas.

So how well could we possibly predict crime?

The random forest model successfully predicted 72% (almost three quarters) of all violent crimes that occurred in 2016. This was superior to the average prediction accuracy of its constituent 1000 decision trees, which was 67% (see Figure 2).

Only 12 out of 1000 individual trees yielded an accuracy better than the random forest. In other words, there is a 99% certainty that predictions from a random forest would be better than that from an individual decision tree.

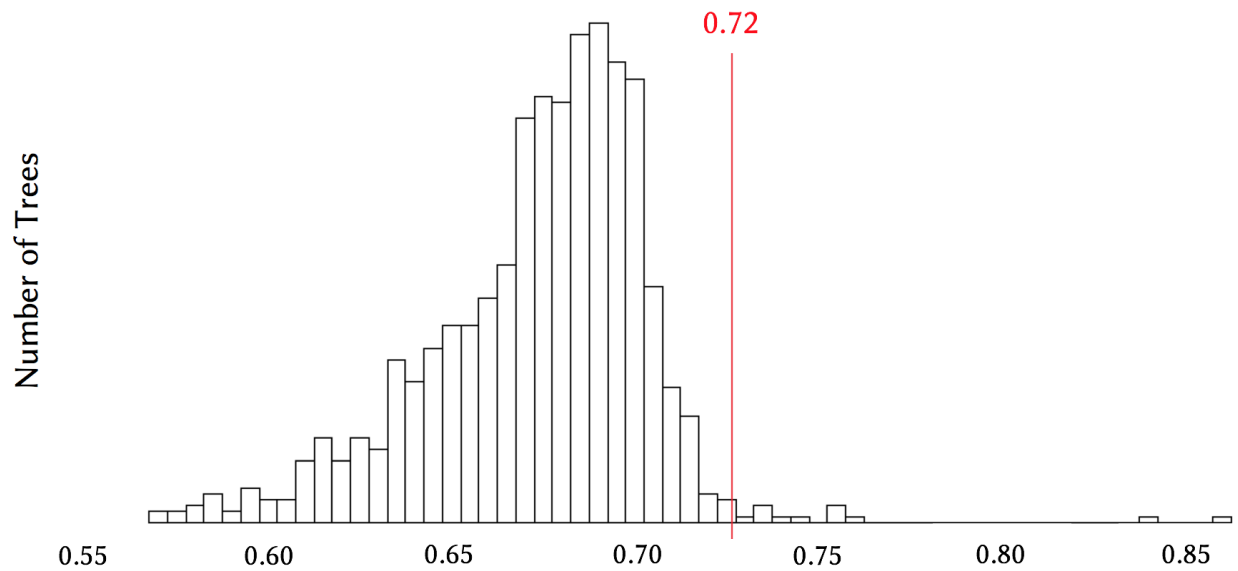


Figure 2. Histogram of prediction accuracies of 1000 decision trees compared against the overall result from combining these trees in a random forest.

Figure 3 shows a sample of the random forest's predictions over 4 days. Based on our predictions, the police should allocate more resources to areas coded red, and fewer to areas coded gray. While it may seem obvious that we need more patrols in areas with historically high crime, the model goes further to pinpoint crime likelihood in non-red areas. For instance, on Day 4, a crime in a gray area (lower-right) was correctly predicted despite no violent crimes occurring there in the prior 3 days.

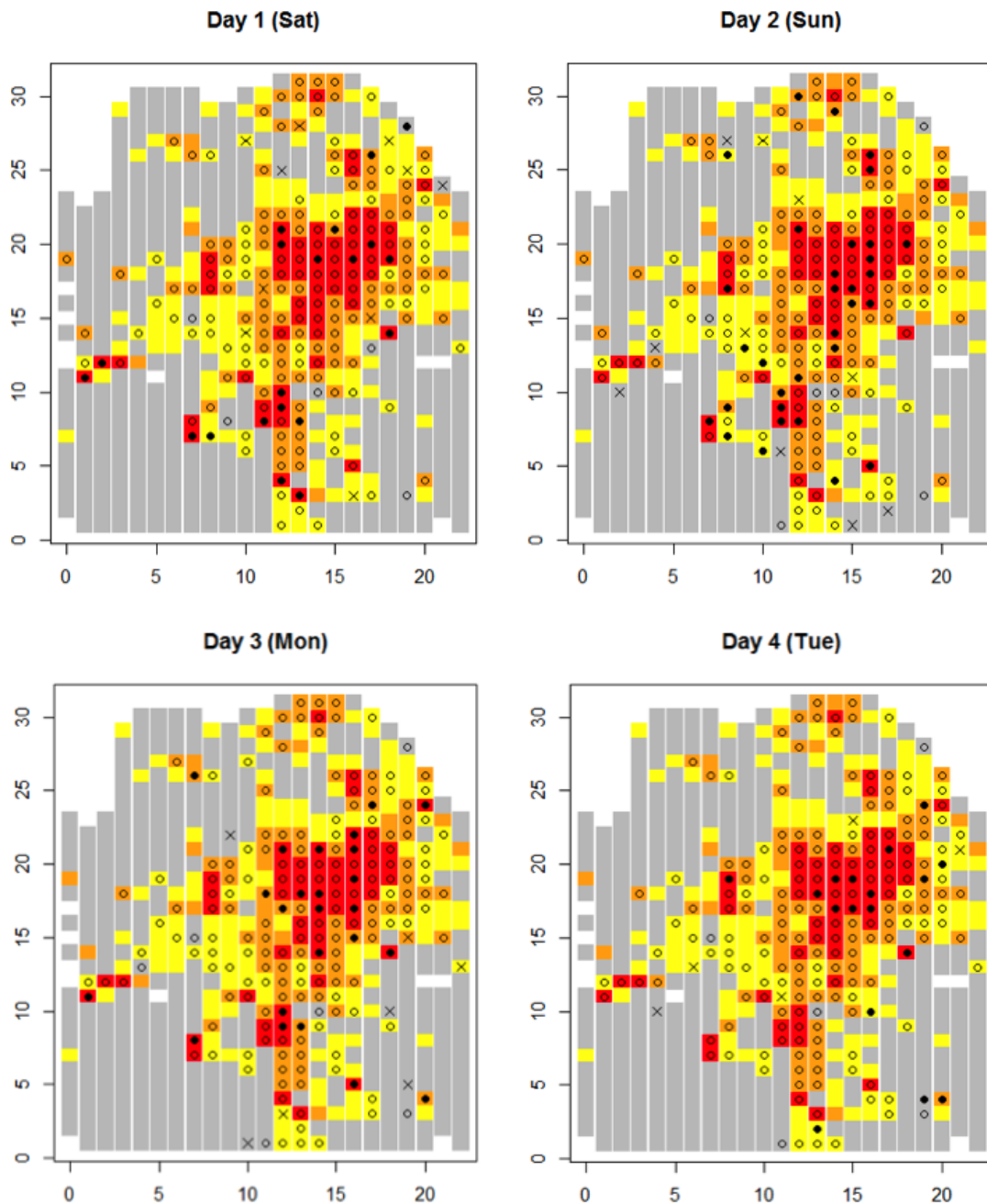


Figure 3. Crime predictions for 4 consecutive days in 2016. Circles denote locations where a violent crime was predicted to happen. Solid circles denote correct predictions. Crosses denote locations where a violent crime happened, but was not predicted by the model.

Besides predicting crime, random forest also allows us to see which variables contribute most to its prediction accuracy. Based on the chart in Figure 4, crime appears to be best forecasted using crime history, location, day of the year and maximum temperature of the day.

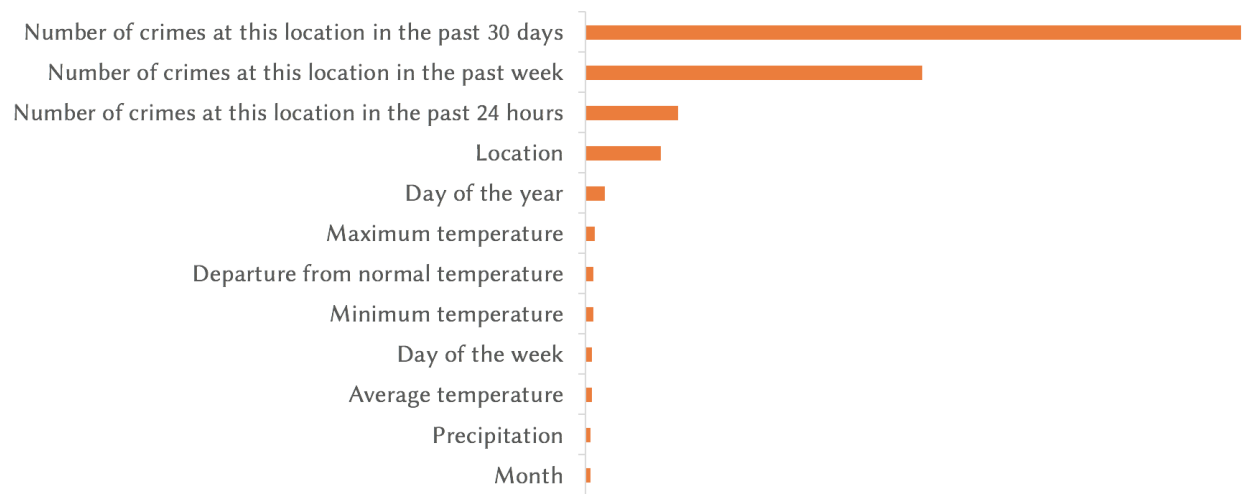


Figure 4. Top variables contributing to the random forest's accuracy in predicting crime.

We have seen how effective random forest could be in predicting a complex phenomenon like crime. But how does random forest work?

Ensembles

A random forest is an ensemble of decision trees; it combines predictions from many different decision trees. In an ensemble, predictions could be combined either by majority-voting or by taking averages. Figure 5 shows how an ensemble formed by majority-voting could yield more accurate predictions than the individual models it is based on.

Model 1	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	7/10 correct
Model 2	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	7/10 correct
Model 3	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	6/10 correct
Ensemble Model (majority voting)	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	8/10 correct

Figure 5. Example of 3 individual models attempting to predict 10 outputs of either Blue or Red. The correct predictions are Blue for all 10 outputs. An ensemble formed by majority voting based on the 3 individual models yields the highest prediction accuracy.

For this effect to work however, models included in the ensemble *must not make the same kind of mistakes*. In other words, the models must be uncorrelated. A systematic way to generate uncorrelated decision trees is to use a technique called *bootstrap aggregating* (bagging).

Bootstrap Aggregating (Bagging)

Recall from our previous chapter that in constructing a [decision tree](#), the dataset is repeatedly divided into subtrees, guided by the best combination of variables. However, finding the right combination of variables can be difficult, as decision trees are prone to [overfitting](#). To overcome this, multiple decision trees could be constructed, by randomizing the combination and order of variables used. We could then use the aggregated result from these trees to form a random forest.

Bootstrap aggregating (bagging) is used to create thousands of decision trees that are sufficiently different from each other. To ensure minimal correlation between trees, each tree is generated from a random subset of the training data, using a random subset of predictor variables. Hence, the trees grown are dissimilar, but they still retain certain predictive power. Figure 6 shows how the predictor variables allowed for selection at each split on a tree are restricted.

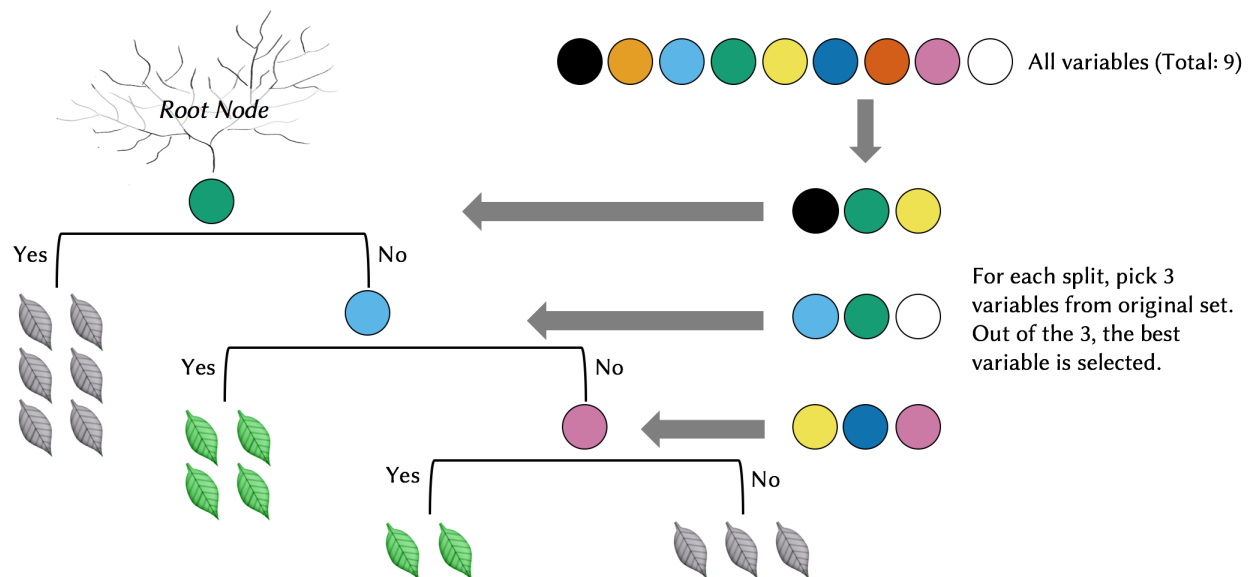


Figure 6. How a tree is created for a random forest.

In Figure 6, there are 9 predictor variables represented by different colors. At each split, a subset of the predictor variables is randomly sampled from the original 9. From each subset, the decision tree algorithm selects the best variable for the split.

By restricting the possible predictors to use at each split in the tree, we are able to generate dissimilar trees that prevent overfitting. To further reduce overfitting, a random forest could be made more complex by increasing the number of trees. The resulting model would thus be more generalizable and accurate.

Limitations

No model is perfect. Choosing whether to use a random forest model is a trade-off between predictive power and interpretability of results.

Not interpretable. Random forests are considered *black boxes*, because they comprise randomly generated decision trees, and are not led by clear prediction rules. For example, we would not know exactly how a random forest model reached its prediction of a crime occurring at a specific place and time. We only know that a majority of its constituent decision trees decided so. The lack of clarity on how predictions are made may bring about ethical concerns when used in areas such as medical diagnosis.

Nonetheless, random forests are widely used because they are easy to implement, especially when accuracy of results is more crucial than interpretability.

A/B Testing and Multi-Armed Bandits

Basics of A/B testing

Imagine that you run an online store, and you wish to put up online advertisements to notify people of an ongoing sale. Which statement would you use?

- Up to 50% discount on items!
- Half-price on selected items

Although the two statements mean the same thing, one could be more persuasive than the other. For instance, you might wish to find out whether it is better to use an exclamation mark to convey excitement, and whether the numerical figure of “50%” is more compelling than the term “half-price”.

To answer these questions, you could display each version of your ad to 100 people, and then check how many of the 100 eventually clicked on your ad to visit your store. The ad that garnered more clicks would likely be more effective in attracting buyers, and thus should be used for the rest of your advertising campaign. This procedure is known as *A/B testing*, in which the effectiveness of ad versions A and B are compared.

Limitations of A/B testing

There are two problems with the method of A/B testing:

- **Results could be a fluke.** A lousier ad could outperform a better ad due to chance, such as buyers being in a happier mood. To be more certain, we could increase the number of people we show each ad version to. However, this leads to a second problem.
- **Loss of potential revenue.** By increasing the number of people to whom we show each ad version from 100 to 200, we would be displaying the lousier ad to more people, potentially losing buyers who might have been persuaded by the better ad.

These two problems represent the tradeoff in A/B testing between *exploration* and *exploitation*. If you increase the number of people to test your ads on (exploration), you could be more certain about which is the better ad, but you would lose more people who might have purchased from your site had they seen the better ad (exploitation).

How do we find balance this trade-off?

Epsilon-Decreasing Strategy

While A/B testing completes an exploration phase for the better ad before committing the rest of the campaign to exploiting that ad, we need not wait for the end of the exploration phase to start exploiting.

If Ad A had garnered more clicks than Ad B amongst the first 100 viewers, we could increase the exposure of Ad A to 60% of the next 100 viewers, while decreasing that of Ad B to 40%. In other words, we could start exploiting the initial result showing Ad A's better performance, but we do not stop exploring a small possibility of improvement in Ad B's performance. As more evidence tilts in Ad A's favor, we could progressively show more of Ad A and less of Ad B.

The above is known as an *epsilon-decreasing strategy*, where *epsilon* refers to the proportion of time you spend exploring an alternative ad to make sure it is indeed less effective. We decrease epsilon as our confidence about which ad is better is reinforced. Hence, this technique falls under a class of machine learning called *reinforcement learning*.

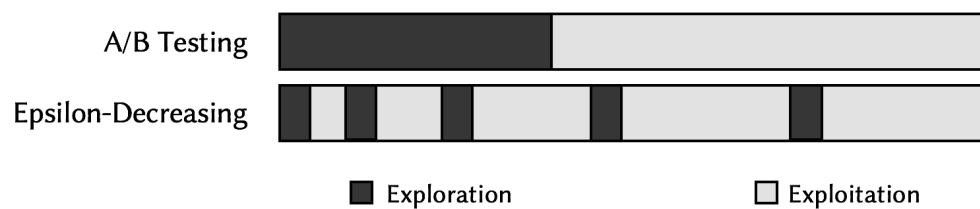


Figure 1. An A/B test comprises one exploration phase followed by one exploitation phase, whereas an epsilon-decreasing strategy intersperses exploration with exploitation, with more exploration at the start and less towards the end.

Example: Multi-Arm Bandits

A typical example used to illustrate the difference between A/B testing and epsilon-decreasing strategy is slot machine play. Slot machines have different rates of payout, and a gambler's aim is to choose which machine to play in order to increase his overall payout.



Figure 2. A one-arm bandit.

As slot machines are often seen as cheating players of money with each arm pull, they have been nicknamed *one-arm bandits*. Having to choose which slot machine to play is thus called a *multi-arm bandit problem*. This term now refers to any problem on how to allocate resources, such as which

slot machine to play, which ad to show, which topics to revise before an exam, or which drug study to fund.

Imagine you have 2 slot machines to choose from, A and B. You have money to play a total of 2000 rounds across both machines. In each round, you pull the arm of one slot machine, which gives you either \$1 or nothing.

Table 1. Machine Payout Rates.

Machine	Payout Rate
A	50%
B	40%

Machine A has a 50% chance of payout, while Machine B has a 40% chance of payout. However, these probabilities of payout are unknown to you.

The question is, what is the best way to play to maximize your winnings?

Total Exploration. If we play each machine randomly, we would get \$900 on average.

A/B Testing. If we use A/B testing to explore which slot machine has a higher payout for the first 200 rounds, and then exploit Machine A for the next 1800 rounds, we would get \$976 on average. There is a catch. Since the payout rates for both machines are similar, there is an 8% chance we might misidentify Machine B as having a higher payout.

To reduce the risk of an identification error, we could extend exploration over 500 rounds. This would result in a 1% chance of misidentifying the higher-paying machine, but it would also decrease the resulting winnings to about \$963.

Epsilon-decreasing Strategy. If we use an epsilon-decreasing strategy to start exploiting a seemingly better machine as we continue exploring to confirm our guess, we could get \$984 on average, while tolerating a 4% chance of misidentification. We could decrease the risk of identification error by increasing the rate of exploration (i.e. value of epsilon), but as before, this would decrease our average winnings.

Total Exploitation. If we had insider knowledge that Machine A had a higher payout, we could exploit it from the very start and get \$1000 on average.

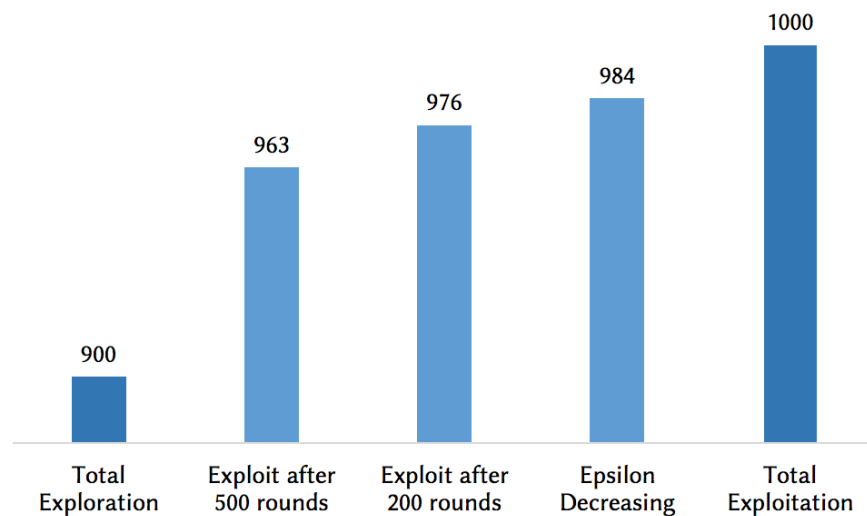


Figure 3. Comparing payouts from different playing strategies.

From Figure 3, it is clear that an epsilon-decreasing strategy would yield the highest winnings. Moreover, for a large number of games, this strategy would surely reveal the better machine due to a mathematical property called convergence.

While an epsilon-decreasing strategy seems superior, it also has limitations that render it more difficult to implement compared to A/B testing.

Limitations of an Epsilon-Decreasing Strategy

To use an epsilon-decreasing strategy, controlling the value of epsilon is crucial. If epsilon decreases too slowly, we could miss out on exploiting the better machine. If it decreases too quickly, we might identify the wrong machine to exploit.

The optimal rate of decrease for epsilon depends heavily on the extent to which payout rates of the two machines differ. If the payout rates are highly similar like in Table 1, epsilon should be decreased slowly. To compute epsilon, a method called *Thompson sampling* could be used.

An epsilon-decreasing strategy also depends on the following assumptions:

1. **Payout rate is constant over time.** An ad might be popular in the morning but not at night, while another ad might be moderately popular throughout the day. If we compare both ads in the morning, we would falsely conclude that the first ad is better.
2. **Payout rate is independent of previous plays.** If an ad is shown to the same customer multiple times, he might grow curious and be more likely to click on it. This means that repeated exploration might be needed to reveal true payouts.
3. **Minimal delay between playing a machine and observing the payout.** If an ad is sent via email, potential buyers might take a few days to respond. This would prevent us from

knowing the true results of our exploration immediately, and any exploitation attempts in the meantime would be based on incomplete information.

Nonetheless, if either the 2nd or 3rd assumption is violated by both products being compared, the effect of any errors could be cancelled out. For instance, if two ads are to be sent via email, any errors in measuring ad performance would be present for both ads, so comparing the ads against each other would still be fair.

While the above assumptions also apply to A/B testing, they are harder to address in an epsilon-decreasing strategy, where more detailed computations for epsilon would be required.

Fun Fact: Sticking to the Winner

An interesting application of the multi-arm bandit problem is in sports. During his tenure with the famed soccer club *Manchester United*, manager Louis van Gaal would adopt an unconventional strategy to decide how his players would take penalty shoots.

The first player appointed to take penalties would continue to do so, until he misses, following which the next player would take over, until he misses again, and so on. This strategy is called *sticking to the winner*.

If we see our slot machines in Table 1 as akin to soccer players with unknown goal-scoring rates, sticking to a machine when we win and switching when we lose would give us an average winning of about \$909, which is only slightly better than playing randomly. By switching frequently, this method explores too much and exploits too little. Furthermore, it overlooks a machine's history of performance and evaluates it solely based on the last game. Hence, it is a less than ideal strategy.