Predicting Hit Songs: A Scikit-Learn Approach

David Wiley
Dr. Vasilis Zafiris

Abstract

To be added.

1. Introduction

The IFPI's (International Federation of the Phonographic Industry) Global Music Report 2019[1] states the total revenue for 2018 was US$19.1 billion. The report also states the global recorded music market grew by 9.7% since 2017. With an increase in recorded music and the potential revenue it generates it is evident that being able to predict whether a song will be a hit or not could prove to be an extremely lucrative practice. The purpose of this project is to explore this practice using data analysis and machine learning methods.

2. Motivation

There are multiple ways to implement and program machine learning. One could program a machine learning model from scratch but there are also various software programs and packages that make life alot easier. The objective of this project is to get use a familiar programming language, Python, and get exposure applying machine learning methods using the Scikit-Learn[2] package on data.

3. Data

The data used in this project was a combination of 55,000+ Song Lyrics[3] dataset downloaded from www.Kaggle.com and a scraped dataset from the Billboard Hot 100[4]. _**"On the week ending November 12, 1955, Billboard published The Top 100 for the first time. The Top 100 combined all aspects of a single's performance (sales, airplay and jukebox activity), based on a point system that typically gave sales (purchases) more weight than radio airplay. Then in 1958, "Billboard premiered one main all-genre singles chart: the Hot 100. The Hot 100 quickly became the industry standard."** The Hot 100 will be used in this project as a measurement of popularity. If the track has appeared on the list, it will be considered as popular and referenced as such throughout this paper. If the track was in the Billboard dataset, it was given a popularity

value of 1 and 0 otherwise. See Figure 3-1 below for an example of what the data looks like.

spotify_id	year	title	artist	peak_pos	popularity	energy	liveness	tempo	speechiness	acousticness	instrumentalness	time_signature	danceability	key	duration_ms	loudness	valence	mode
000xQL6tZNLJzIrtIgxqSl	2017	Still Got Time	zayn	66	1	0.627	0.0852	120.963	0.0644	0.131	0	4	0.748	7	188491	-6.029	0.524	1
0010mZpCCwlPwoBiBsjoF	2014	It's Alright Now	Bombay Bicycle Club	0	0	0.793	0.144	124.994	0.0352	0.108	3.28E-06	3	0.527	3	249947	-4.823	0.597	1
002opcRBgYV5jqoh72QcP	2016	Just to See You Smile	Why Don't We	0	0	0.618	0.102	89.901	0.142	0.0508	0	4	0.707	9	181958	-7.154	0.261	1
003VWU5LKbDkXTqpLQP	2013	The Outsiders	eric church	51	1	0.808	0.101	80.026	0.0633	0.282	4.97E-05	4	0.521	7	253640	-5.204	0.323	1
005lwxGU1tms6HGELIcLV	2008	I Kissed A Girl	katy perry	1	1	0.76	0.132	129.996	0.0677	0.00223	0	4	0.699	5	179640	-3.173	0.696	1
006hfStQk9L7MVT9RcDd	2014	A Part Of Me (feat. Laura Whit	Neck Deep	0	0	0.875	0.103	100.018	0.0435	0.376	5.16E-05	4	0.372	4	189415	-0.946	0.313	1
006x3wfSReHJcoRc3LpO	2010	More Lost Without You - Live	Alan Parsons	0	0	0.639	0.77	96.076	0.0327	0.643	0.000279	4	0.584	2	205427	-8.25	0.701	1
007n10xGvSbc7dKgAOR	2003	Brokenheartsville	joe nichols	27	1	0.65	0.0956	103.954	0.0228	0.142	0.000325	4	0.668	10	231347	-4.351	0.577	1
00BNkn0PppbR5dEOTcA	2015	Mr Mambo (Hips Mix)	11 Acorn Lane	0	0	0.95	0.0476	127.991	0.0381	0.188	0.0938	4	0.909	0	186941	-4.795	0.968	1
00dhuugTq7XaBotK71vM	2009	Sweet Caroline	glee cast	34	1	0.449	0.0792	124.035	0.0337	0.07	6.91E-05	4	0.63	11	118133	-7.127	0.638	1
00dssbjaUrePKZcYBN5oA	2017	Oh, How I Love Jesus	Reba McEntire	0	0	0.42	0.0869	93.77	0.027	0.73	0	3	0.257	7	198160	-6.617	0.233	1
00eyIJu8FmiibWsppUVOP	2015	Jesus, Don't Turn Me Away	Vern Gosdin	0	0	0.258	0.125	77.744	0.0292	0.602	1.74E-06	4	0.717	2	178480	-15.547	0.656	1
00F7eT1SZeNJGiXp70AM	2015	Thinking Like a Living Legend	Moxkito	0	0	0.847	0.338	192.742	0.281	0.747	0	4	0.469	5	234031	-5.657	0.755	1

Figure 3-1. An example of the csv file that includes a few observations of the data used in the project.

3.1

The popular data was scraped from the Billboard Hot 100 website using Python methods in Jupyter. Since the Hot 100 list is released every week on Saturday, the dates focused start from January, 1, 2000. This specific date was used because musical styles and preferences change throughout the years, so we did not want to reference songs that were a little "outdated" in comparison to today's musical styles. The latest date used is December 29, 2018, the last Saturday of that year. This is because our unpopular data, the dataset from Kaggle, only extends to the year 2018. The data extracted includes: track name, artist name, current position for that week, position the previous week, highest position it has reached, weeks it has been on the chart, the year of the position week, month of the position week, and day of the week. Since songs can stay on the chart for multiple weeks at a time and our objective does not concern when the song is on the chart, only that it is on the chart, duplicates were removed without regard of week. After removing duplicates, there were ~7,800 songs.

3.2

The unpopular data is pulled from Kaggle. This dataset has 57,650 songs where each include: artist name, track name, and lyrics. The data was designed for Natural Language Processing analysis, so it includes the lyrics for those songs and this specific data was deleted because it is not needed for our purposes. To get the dates for these songs we used the `spotipy`[5] package to search and populate each song with its release date. The songs were then filtered for the years 2000-2018. This left over 15,000 songs in the unpopular dataset. Using Python methods, songs were selected random, with an even amount from each year, to match the size of the popular dataset.

3.3

The features that will be used in our model will be extracted from Spotify using the `spotipy` package in Python. The table and definitions for the measurements used for these features are provided by Spotify[6].

Table 1

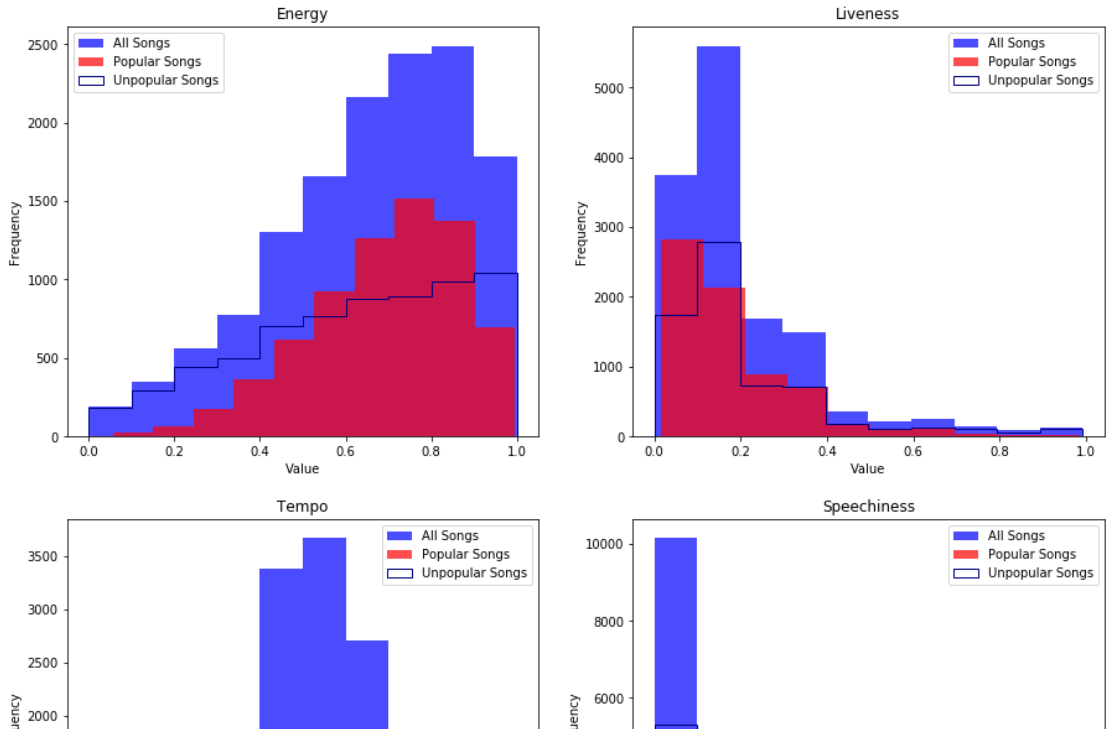
Audio Features Object

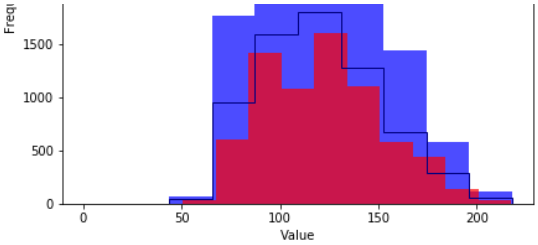
Key	Value Type	Description
energy	*float*	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
liveness	*float*	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
tempo	*float*	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
speechiness	*float*	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.

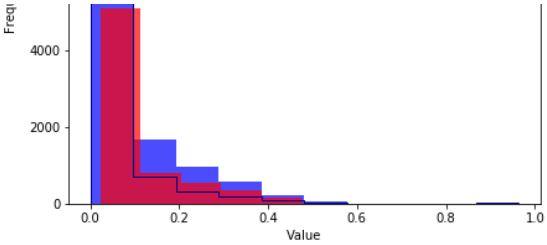
Key	Value Type	Description
acousticness	*float*	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
instrumentalness	*float*	Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
time_signature	*int*	An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
danceability	*float*	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
key	*int*	The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation . E.g. 0 = C, 1 = C#/Db, 2 = D, and so on. If no key was detected, the value is -1.
duration_ms	*int*	The duration of the track in milliseconds.

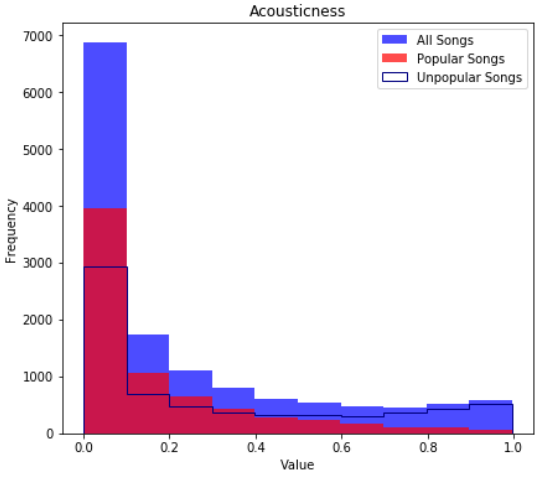
Key	Value Type	Description
loudness	*float*	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.
valence	*float*	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
mode	*int*	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.

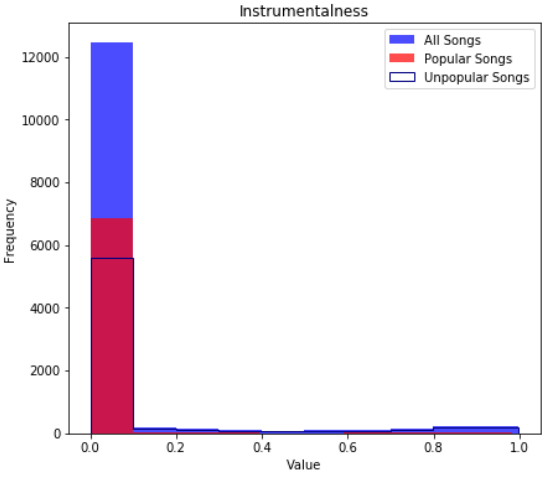
4. Data Analysis

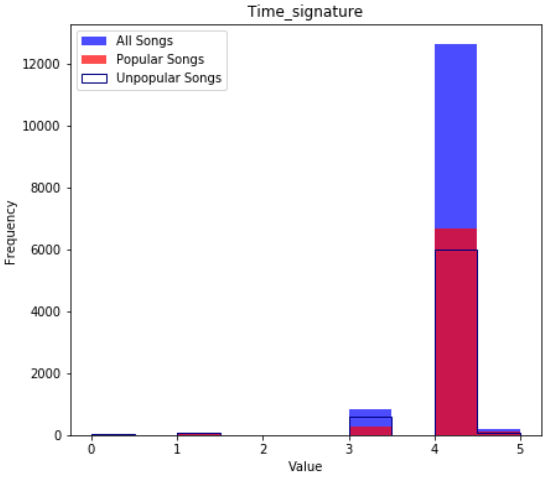


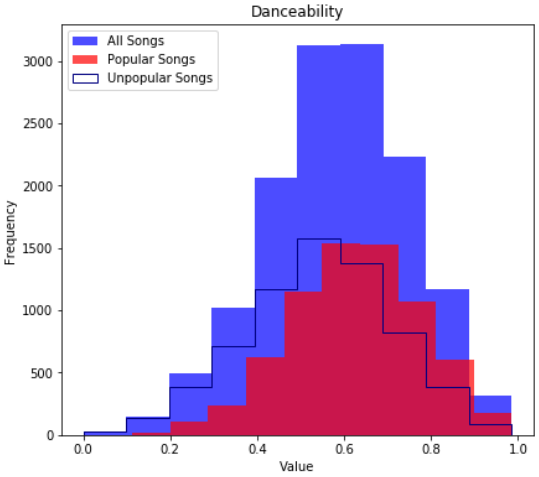

Value

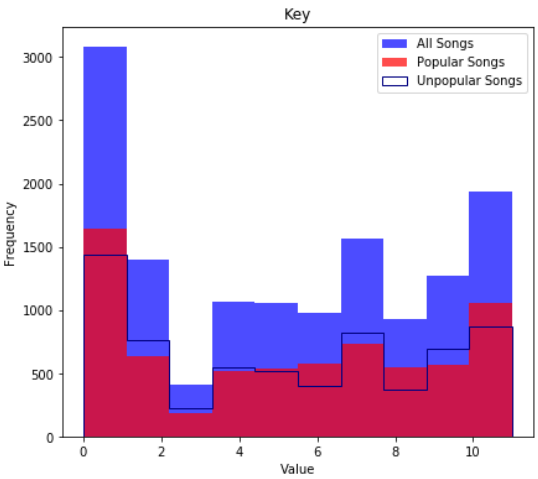

Value

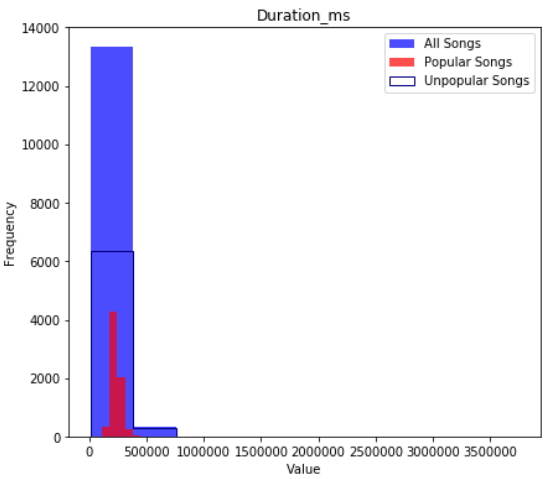

Acousticness
All Songs
Popular Songs
Unpopular Songs
Frequency
Value

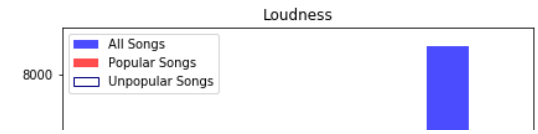

Instrumentalness
All Songs
Popular Songs
Unpopular Songs
Frequency
Value

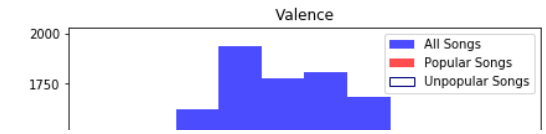

Time_signature
All Songs
Popular Songs
Unpopular Songs
Frequency
Value

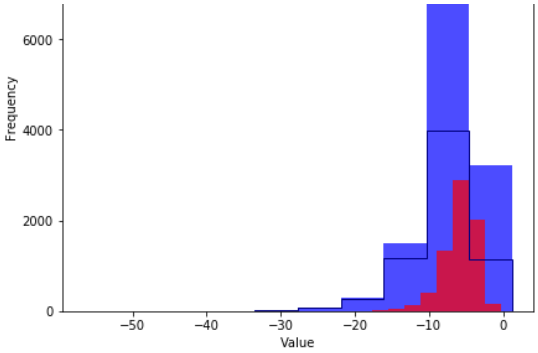

Danceability
All Songs
Popular Songs
Unpopular Songs
Frequency
Value

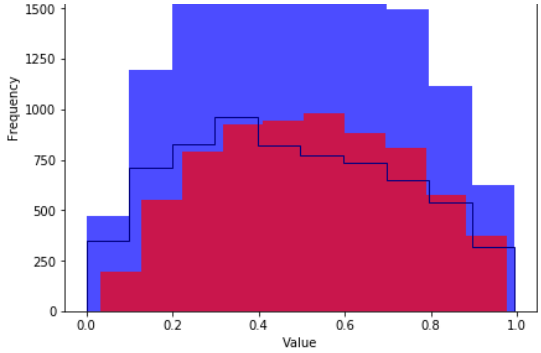

Key
All Songs
Popular Songs
Unpopular Songs
Frequency
Value

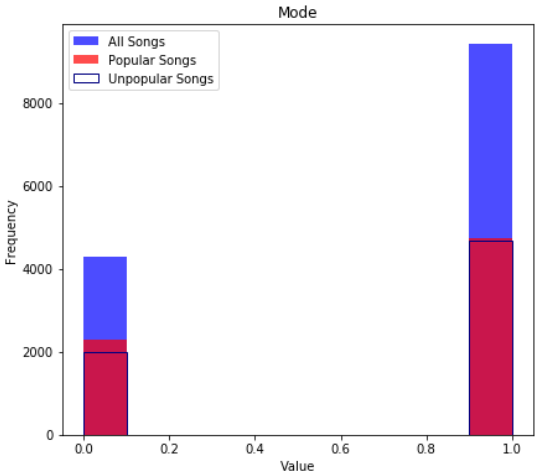

Duration_ms
All Songs
Popular Songs
Unpopular Songs
Frequency
Value


Loudness
All Songs
Popular Songs
Unpopular Songs


Valence
All Songs
Popular Songs
Unpopular Songs







Analysis of the data will be here.

How is the data train/test split?

5. Methods

Overview

There are numerous types of models the Scikit-Learn package offers that can be applied which allows users to choose the best model for their data. In this project, we will be using the *supervised learning* method because we have a known dependent variable, popularity, in our data set. More specifically, the *classifying* method will be used because we are classifying each song as popular or unpopular. Since we will be classifying a song, we will be utilizing the Logistic Regression, Stochastic Gradient Descent Classifier, k-Nearest Neighbors, Random Forest Classifier, and Discriminant Analysis classifying models. Each model has multiple parameters that can be changed and adjusted to fine tune the model to fit best with the data and get the desired results.

5.1 - Logistic Regression

The underlying function in Logistic Regression is a Linear Regression function. *Linear Regression* takes the weighted sum of input features from data and returns a value. The *Logistic Regression* takes that weighted sum and passes it through another function, the

Sigmoid function. In Hands-On Machine Learning with Scikit-Learn and Tensorflow[7], Geron uses the Sigmoid function as a function of t, as seen below in Figure 5-1:

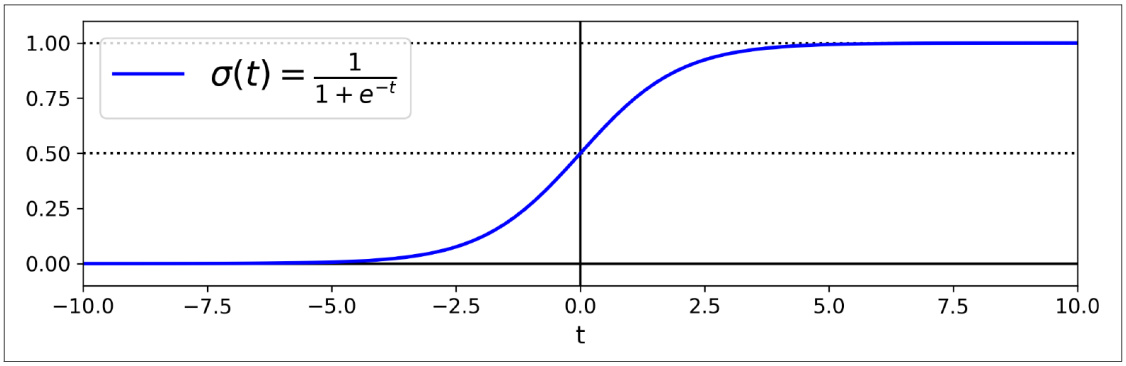


Figure 5-1. Logistic function in Geron's book.

This allows the Logistic Regression to model the probability of an event being a success or failure, 1 or 0 respectively, based on the features in the data. In this case we will be modeling the probability whether a song will be a hit based on the characteristics of that song. Thus, we are predicting the binary response variable, song popularity, based off the effect of the audio features of the song, provided by Spotify.

5.2 - Stochastic Gradient Descent

The basic idea behind Stochastic Gradient Descent is *Gradient Descent*. Gradient Descent is an optimization algorithm that iteratively adjusts the parameters within a function until it finds the minimum of the cost function. This is achieved by calculating the partial derivative of the cost function for the whole data set. Note that the main parameter to be adjusted within a gradient descent is the *learning rate*. The learning rate is the step size taken when descending toward the minimum (see Figure 5-2). However, using Gradient Descent can very computationally costly if the data set is very large. Stochastic Gradient Descent can be used to cut down on time and computational cost.

In *Stochastic Gradient Descent*, a random instance of the data set is used to compute the descent. This will be more efficient than calculating the descent for the whole data set. Also, within Scikit-Learn the step size is initially chosen at random and the algorithm will begin there and update the descent as it iterates. Scikit-Learn also takes a parameter of the maximum iterations to complete until it finishes. By default this is set to 1000 iterations but can be changed by the user.

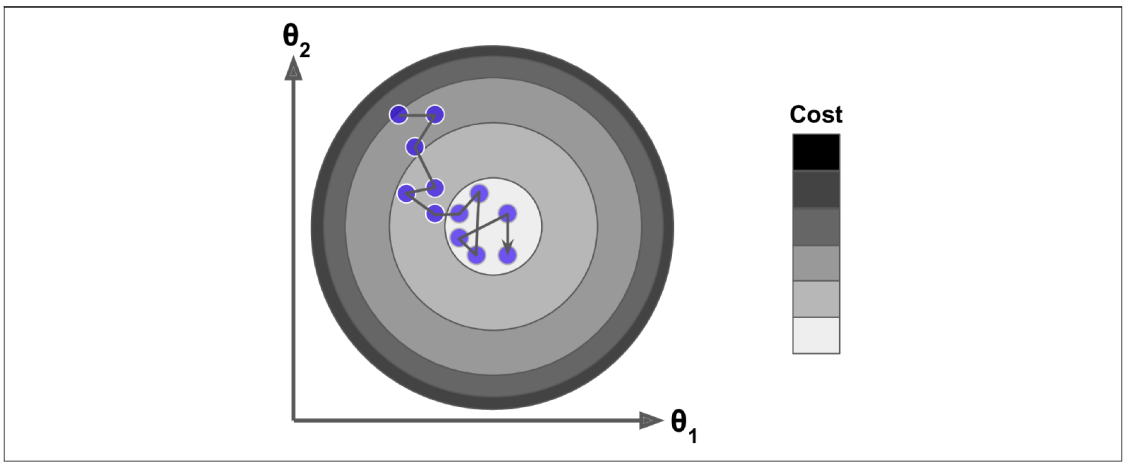


Figure 5-2. Stochastic Gradient Descent pictured in Geron's book.

5.3 - k-Nearest Neighbors

In the *k-Nearest Neighbors* model, the user selects a value, *k*, so that the algorithm will know how many *neighbors*, or data points, to compare to the data point being predicted. Then the distance is calculated between each data point and the predicted data point. The distances are essentially sorted in increasing order and the model sets the class of the predicted data point to the most common class of the first *k* number of neighbors. For example, as shown in Figure 5-3[7], if we wanted to classify the green circle using $k=3$, we would classify the circle to be a triangle. However, if we choose $k=5$ or $k=11$, we would classify the circle to be a square.

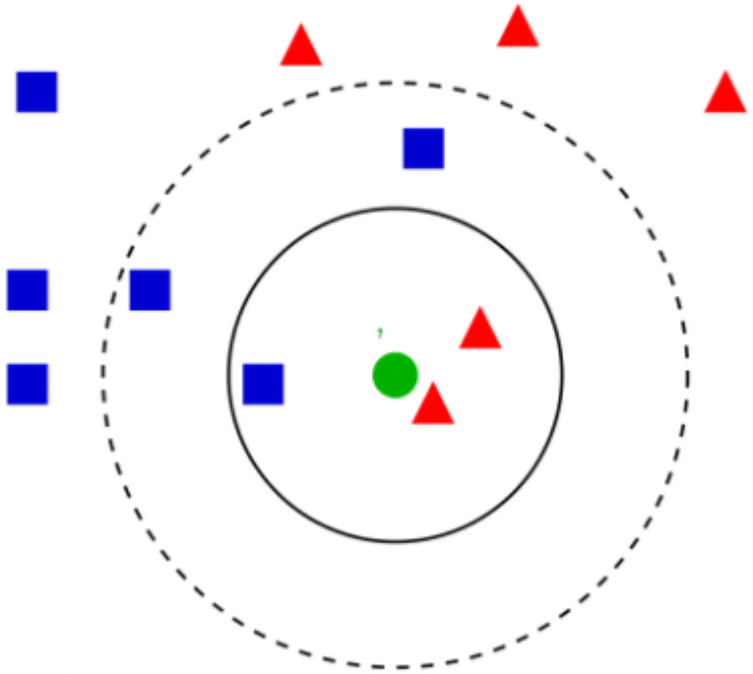


Figure 5-3. A visualization of k-Nearest Neighbors.

5.4 - Random Forest Classifier

The *Random Forest Classifier* in an ensemble of multiple Decision Tree Classifiers. A *Decision Tree Classifier* divides the data space into separate class identifying groups. One can imagine a Decision Tree as a flowchart of decisions of certain criteria. The issue with Decision Trees is due to their simplicity they can lead to overfitting. Therefore a Random Forest Classifier solves this issue by combining multiple Decision Trees and decides on the final class of the data point being predicted.

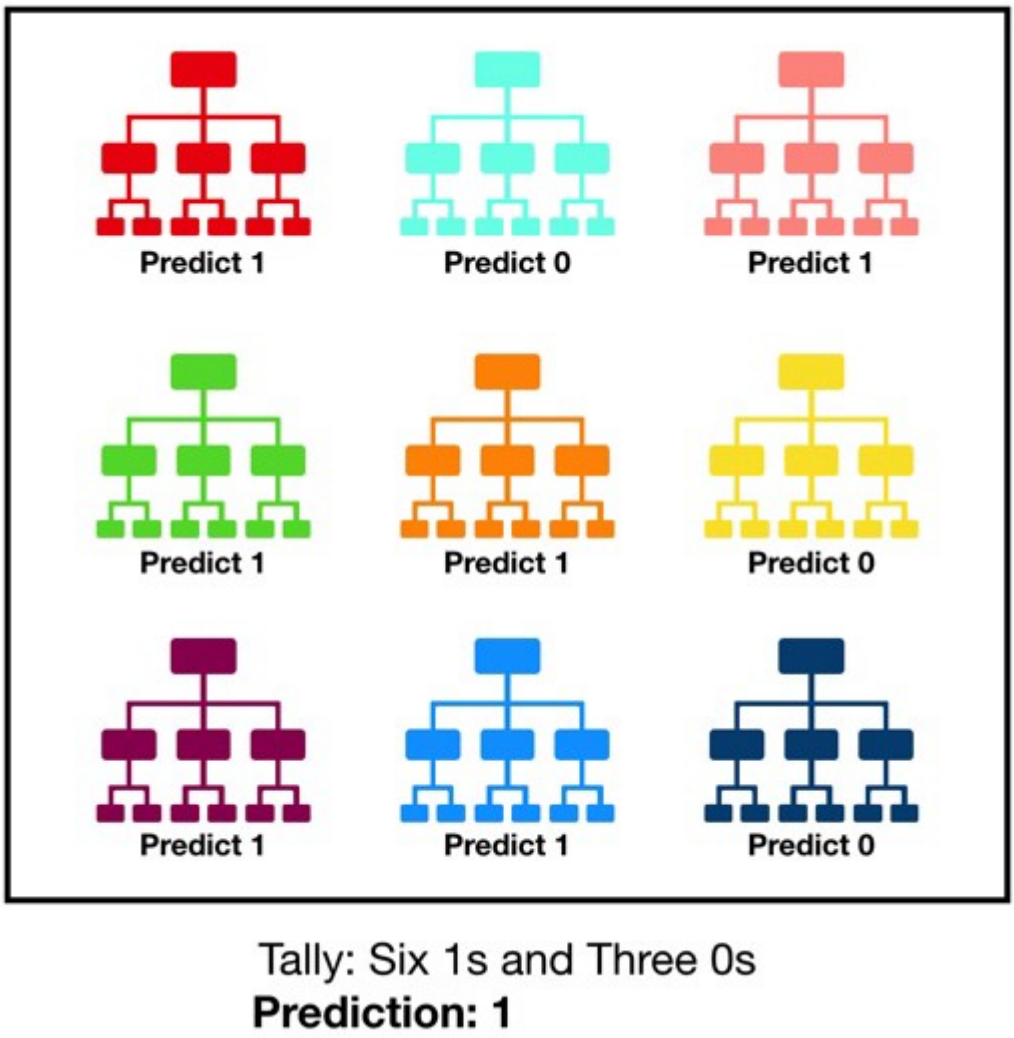


Figure 5-4. A visualization of a Random Forest.

5.5 - Discriminant Analysis

As explained in Geron's book, *Discriminant Analysis* takes the most descriptive features of the training set data and projects it onto a hyperplane defined by those features.

5.6 - Model Parameters

Loss Functions

Loss functions measure how far off the predicted dependent value is from the actual value in the data set. Sometimes these functions are also referred to as *cost* functions. In the Scikit-Learn documentation, they use *penalty* functions. If the loss function produces a large value, it indicates we have a poor prediction and a smaller values indicate we are close to the desired output. The idea is to minimize the loss function as much as possible. According to the documentation, Scikit-Learn uses three different functions:

The $l2$ penalty minimizes:

$min_{w,c}\frac{1}{2}w^{T}w+C\sum_{i=1}^{n}log(exp(-y_{i}(X_{i}^{T}w+c))+1)$

The $l2$ minimizes:

$min_{w,c}||w||*{1}*+C*sum*{i=1}^{n}log(exp(-y_{i}(X_{i}^{T}w+c))+1)$

The elastic-net minimizes:

$min_{w,c}\frac{1-\rho}{2}w^{T}w+\rho||w||*{1}*+C*sum*{i=1}^{n}log(exp(-y(X_{i}^{T}w+c))+1)$

Solvers

There are multiple solvers the Scikit-Learn package uses. These solvers tries to optimize the parameter weights that minimize the cost function: newton-cg, lbfgs, liblinear, sag, and saga.[10]

6. Results and Discussion

Explain the results.

Discuss issues and possible solutions.

7. Conclusion

Conlusion.

8. References

[1] International Federation of the Phonographic Industry (2019, April 2). *IFPI Global Music Report 2019*. Retrieved from https://www.ifpi.org/news/IFPI-GLOBAL-MUSIC-REPORT-2019

[2] Pedregosa et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research(12)*, pp. 2825-2830. Retrieved from

http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

[3] Kuznetsov, S. (2016). *55000+ Song Lyrics* (1) [CSV file with songs, artists and lyrics]. Retrieved from https://www.kaggle.com/mousehead/songlyrics

[4] Top 100 Songs: Billboard Hot 100 Chart (2000-2019). Retrieved from https://www.billboard.com/charts/hot-100

[5] Lamere et al. (2014). *Welcome to Spotipy!*. Retrieved from https://spotipy.readthedocs.io/en/latest/

[6] Spotify (2019). *Audio Features Object*. Retrieved from https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/

[7] Geron, A. (2019). *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. Sebastopol, CA: O'Reilly Media, Inc.

[8] Srivastava, T. (2018, March 26). *Introduction to k-Nearest Neighbors: A powerful Machine Learning Algorithm (with implementation in Python & R)*. Retrieved from https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

[9] Yiu, T. (2019, June 12). *Understanding Random Forest - Towards Data Science*. Retrieved from https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[10] Hale, J. (2019, October 18). *Don't Sweat the Solver Stuff - Towards Data Science*. Retrieved from https://towardsdatascience.com/dont-sweat-the-solver-stuff-aea7cddc3451