Spotify Genre Classification Algorithm

Supervised Machine Learning — SVM, Random Forest, Logistic Regression

Cd · Follow
Published in Towards Data Science
12 min read · Apr 23, 2021

Listen Share

This article assumes a basic understanding of machine learning algorithms and data science techniques.

Article outline:

- Supervised machine learning
- Classification — multi-class
- Dataset — preliminary analysis & feature selection
- Algorithm selection — SVM, Logistic Regression, Random Forest
- Model performance — accuracy scores
- Improvements — hyperparameter tuning & ensemble learning
- Conclusions — more data!

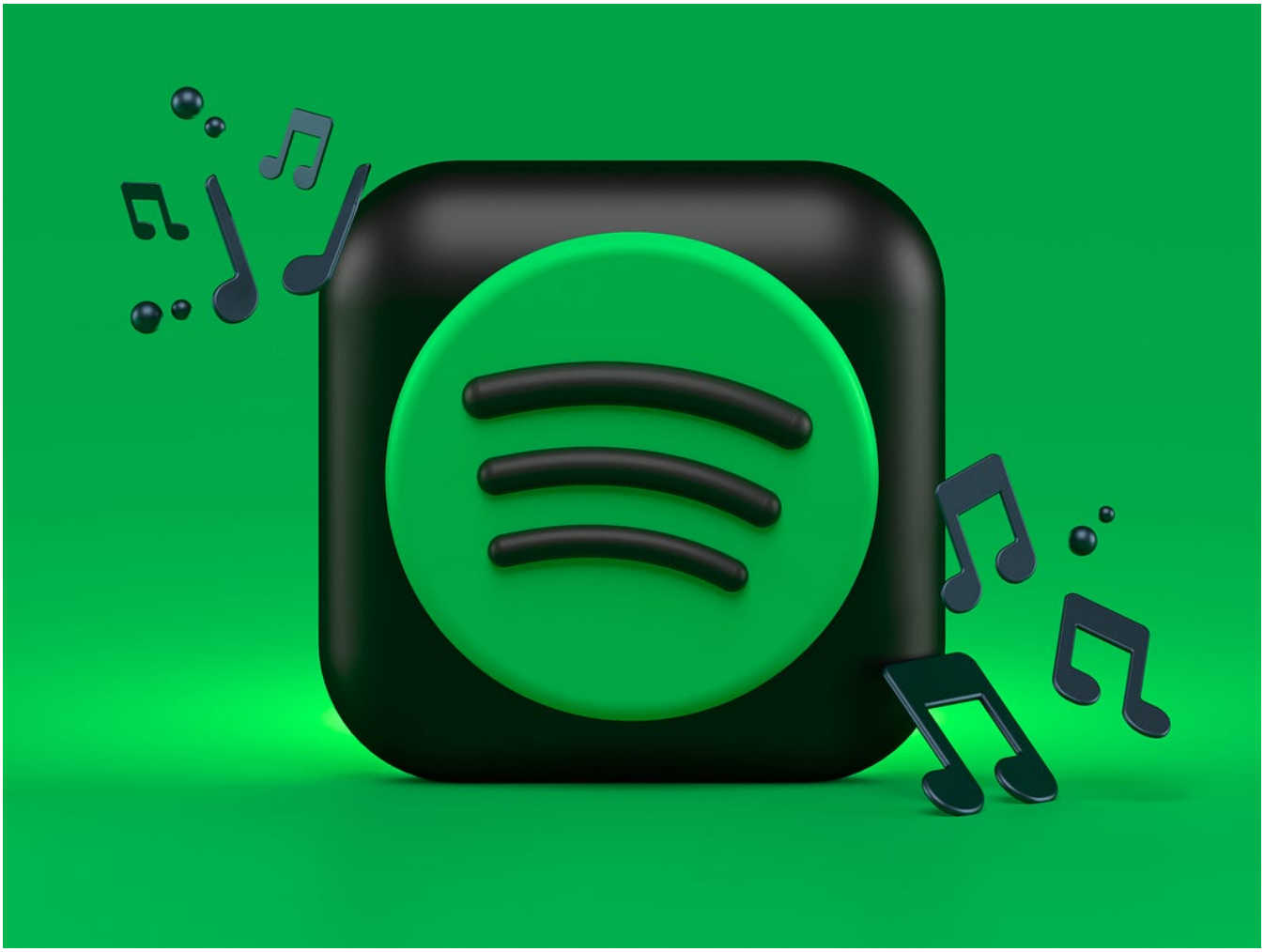


Photo by Alexander Shatov on Unsplash

What is Supervised Machine Learning?

As with all technologies there are buzzwords, supervised learning is an umbrella term to describe an area of machine learning (the most frequently used in practice) where the data being used is **labelled.** The goal of a supervised learning algorithm is to leverage a dataset to produce a model that takes a feature vector (x) as input and outputs variable (Y). An algorithm is used to learn the mapping function from input to output, this is then used with new unseen input data to predict the output variables for that data.

What is Classification?

A classification algorithm takes a dataset of labelled examples as inputs to produce a model that can take unlabeled new data and automatically assign labels to the unlabeled example.

If the classification problem has a set of two labels (for instance “spam” or “not spam”) then it is a binary classification problem. Multi-class classification is a problem where the number of labels within the set is three or greater. The problem that we are looking at is a multi-class as there are many genres within the set.

Show me the Data

The dataset being examined is a collection of song information. It is available on Kaggle and Github alongside my python code.

The data has been split into labelled training data and unlabeled test data.

	Id	title	artist	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	pop	top genre
0	1	My Happiness	Connie Francis	1996	107	31	45	-8	13	28	150	75	3	44	adult standards
1	2	Unchained Melody	The Teddy Bears	2011	114	44	53	-8	13	47	139	49	3	37	NaN
2	3	How Deep Is Your Love	Bee Gees	1979	105	36	63	-9	13	67	245	11	3	77	adult standards
3	4	Woman in Love	Barbra Streisand	1980	170	28	47	-16	13	33	232	25	3	67	adult standards
4	5	Goodbye Yellow Brick Road - Remastered 2014	Elton John	1973	121	47	56	-8	15	40	193	45	3	63	glam rock

Image by Author (Labelled Training Data)

	Id	title	artist	year	bpm	nrgy	dnce	dB	live	val	dur	acous	spch	pop
0	454	Pump It	The Black Eyed Peas	2005	154	93	65	-3	75	74	213	1	18	72
1	455	Circle of Life - From "The Lion King"/Soundtra...	Elton John	1994	161	39	30	-15	11	14	292	26	3	59
2	456	We Are The Champions - Remastered 2011	Queen	1977	64	46	27	-7	12	18	179	38	3	76
3	457	Insomnia - Radio Edit	Faithless	2010	127	92	71	-9	37	53	216	6	4	50
4	458	This Eve of Parting	John Hartford	2018	115	46	56	-12	21	34	153	18	3	44

Image by Author (Unlabeled Test Data)

- Id — an arbitrary unique track identifier
- title — track title
- artist — singer or band
- year — year of release (or re-release)
- bpm — beats per minute (tempo)
- nrgy — energy: the higher the value the more energetic
- dnce — danceability: the higher the value, the easier it is to dance to this song
- dB — loudness (dB): the higher the value, the louder the song
- live — liveness: the higher the value, the more likely the song is a live recording
- val — valence: the higher the value, the more positive mood for the song
- dur — duration: the length of the song

- acous — acousticness: the higher the value the more acoustic the song is
- spch — speechiness: the higher the value the more spoken word the song contains
- pop — popularity: the higher the value the more popular the song is
- top genre — genre of the track (and the target variable for this problem)

There were 15 null values identified within the training set within the top_genre column. The missingo library provides a great visualization for missing values, making it easy to identify the columns that have null values. The 15 null values were dropped.

```
import missingno as msno
msno.bar(class_train, color="dodgerblue", sort="ascending", figsize=
(10,5), fontsize=12)
class_train["top genre"].isnull().value_counts()
# dropping NULL values
class_train = class_train.dropna(axis=0)
```

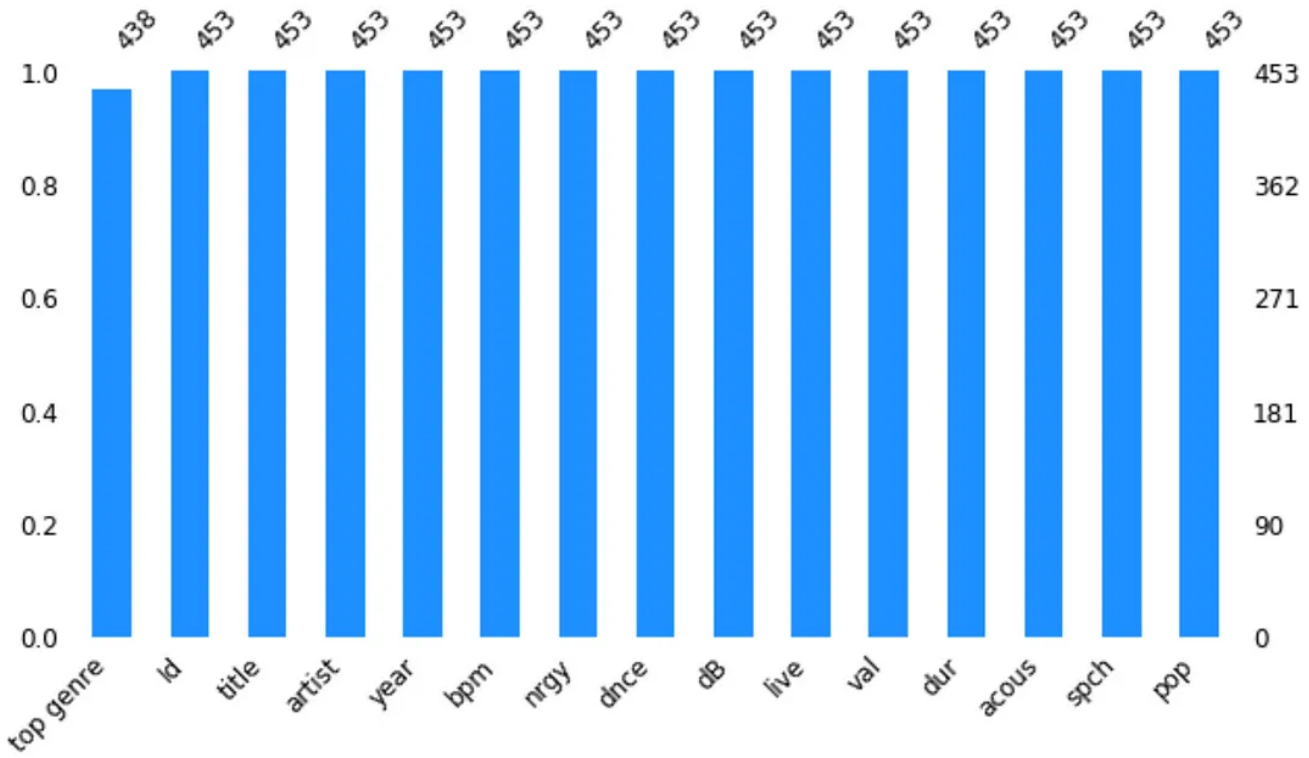


Image by Author

Prior to any model creation it is good practice to check for multicollinearity, which is correlation between the independent features within the dataset. The easiest way to check this is with a correlation heatmap. It is clear there is no multicollinearity.

```
# Plot linear correlation matrix
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(class_train.corr(), annot=True, cmap='YlGnBu', vmin=-1,
vmax=1, center=0, ax=ax)
plt.title('LINEAR CORRELATION MATRIX - CLASS_TRAIN')
plt.show()
```

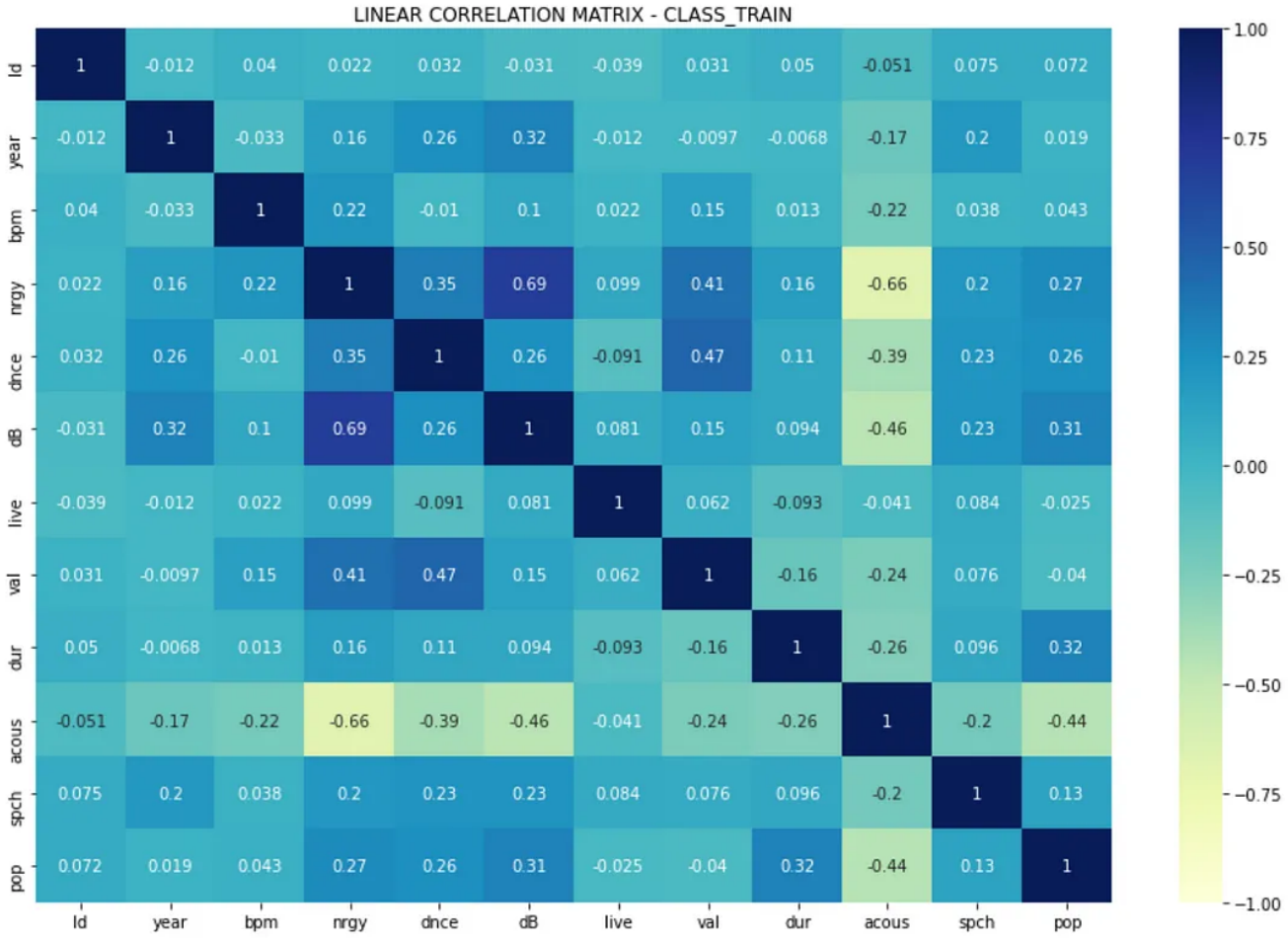


Image by Author

The pop (popularity) column can be used to create another column that signifies if a song is liked (1) or not (2). Histograms can then be used to display the “Like” distribution for each of the features in the dataset. This is an example of feature engineering.

```
conditions = [
    (class_train['pop'] >= 55),
    (class_train['pop'] < 55) ]
values = [1, 2]
class_train['like'] = np.select(conditions, values)

# for all features
pos_bpm = class_train[class_train['like'] == 1]['bpm']
neg_bpm = class_train[class_train['like'] == 2]['bpm']
pos_nrgy = class_train[class_train['like'] == 1]['nrgy']
neg_nrgy = class_train[class_train['like'] == 2]['nrgy']
pos_db = class_train[class_train['like'] == 1]['dB']
neg_db = class_train[class_train['like'] == 2]['dB']
pos_live = class_train[class_train['like'] == 1]['live']
neg_live = class_train[class_train['like'] == 2]['live']
pos_dur = class_train[class_train['like'] == 1]['dur']
neg_dur = class_train[class_train['like'] == 2]['dur']
pos_acous = class_train[class_train['like'] == 1]['acous']
neg_acous = class_train[class_train['like'] == 2]['acous']
pos_spch = class_train[class_train['like'] == 1]['spch']
neg_spch = class_train[class_train['like'] == 2]['spch']
pos_val = class_train[class_train['like'] == 1]['val']
neg_val = class_train[class_train['like'] == 2]['val']
pos_dnce = class_train[class_train['like'] == 1]['dnce']
neg_dnce = class_train[class_train['like'] == 2]['dnce']

fig2 = plt.figure(figsize=(20,20))
#dnce
ax3 = fig2.add_subplot(331)
ax3.set_xlabel('Danceability')
ax3.set_ylabel('Count')
ax3.set_title('Song Danceability Like Distribution')

pos_dnce.hist(alpha=0.5, bins=30)
ax4 = fig2.add_subplot(331)

neg_dnce.hist(alpha=0.5, bins=30)
plt.legend(['Like', 'Dislike'])

#duration
ax5 = fig2.add_subplot(332)
ax5.set_xlabel('Duration')
ax5.set_ylabel('Count')
ax5.set_title('Song Duration Like Distribution')

pos_dur.hist(alpha=0.5, bins=30)
ax6 = fig2.add_subplot(332)

neg_dur.hist(alpha=0.5, bins=30)
plt.legend(['Like', 'Dislike'])

# loudness (dB)
ax7 = fig2.add_subplot(333)
ax7.set_xlabel('Loudness -dB')
ax7.set_ylabel('Count')
```

```
ax7.set_title('Song Loudness Like Distribution')
plt.legend(['Like', 'Dislike'])

pos_db.hist(alpha=0.5, bins=30)
ax8 = fig2.add_subplot(333)

neg_db.hist(alpha=0.5, bins=30)
plt.legend(['Like', 'Dislike'])

# energy
ax9 = fig2.add_subplot(334)
ax9.set_xlabel('Energy')
ax9.set_ylabel('Count')
ax9.set_title('Song Energy Like Distribution')

pos_nrgy.hist(alpha=0.5, bins=30)
ax9 = fig2.add_subplot(334)

neg_nrgy.hist(alpha=0.5, bins=30)
plt.legend(['Like', 'Dislike'])

# live
ax10 = fig2.add_subplot(335)
ax10.set_xlabel('Liveness')
ax10.set_ylabel('Count')
ax10.set_title('Liveness - Like Distribution')

pos_live.hist(alpha=0.5, bins=30)
ax11 = fig2.add_subplot(335)

neg_live.hist(alpha=0.5, bins=30)
plt.legend(['Like', 'Dislike'])

# val
ax12 = fig2.add_subplot(336)
ax12.set_xlabel('Valence')
ax12.set_ylabel('Count')
ax12.set_title('Valence (Mood?) - Like Distribution')

pos_val.hist(alpha=0.5, bins=30)
ax13 = fig2.add_subplot(336)

neg_val.hist(alpha=0.5, bins=30)
plt.legend(['Like', 'Dislike'])

# acous
ax14 = fig2.add_subplot(337)
ax14.set_xlabel('Acousticness')
ax14.set_ylabel('Count')
ax14.set_title('Acousticness - Like Distribution')

pos_acous.hist(alpha=0.5, bins=30)
ax15 = fig2.add_subplot(337)

neg_acous.hist(alpha=0.5, bins=30)
plt.legend(['Like', 'Dislike'])

# speech
ax16 = fig2.add_subplot(338)
```

```
ax16.set_xlabel('Speech')
ax16.set_ylabel('Count')
ax16.set_title('Speech - Like Distribution')

pos_spch.hist(alpha=0.5, bins=30)
ax17 = fig2.add_subplot(338)

neg_spch.hist(alpha=0.5, bins=30)
plt.legend(['Like', 'Dislike'])

# bpm
ax18 = fig2.add_subplot(339)
ax18.set_xlabel('Beats Per Minute')
ax18.set_ylabel('Count')
ax18.set_title('Song BPM - Like Distribution')

pos_bpm.hist(alpha=0.5, bins=30)
ax19 = fig2.add_subplot(339)

neg_bpm.hist(alpha=0.5, bins=30)
plt.legend(['Like', 'Dislike'])
```

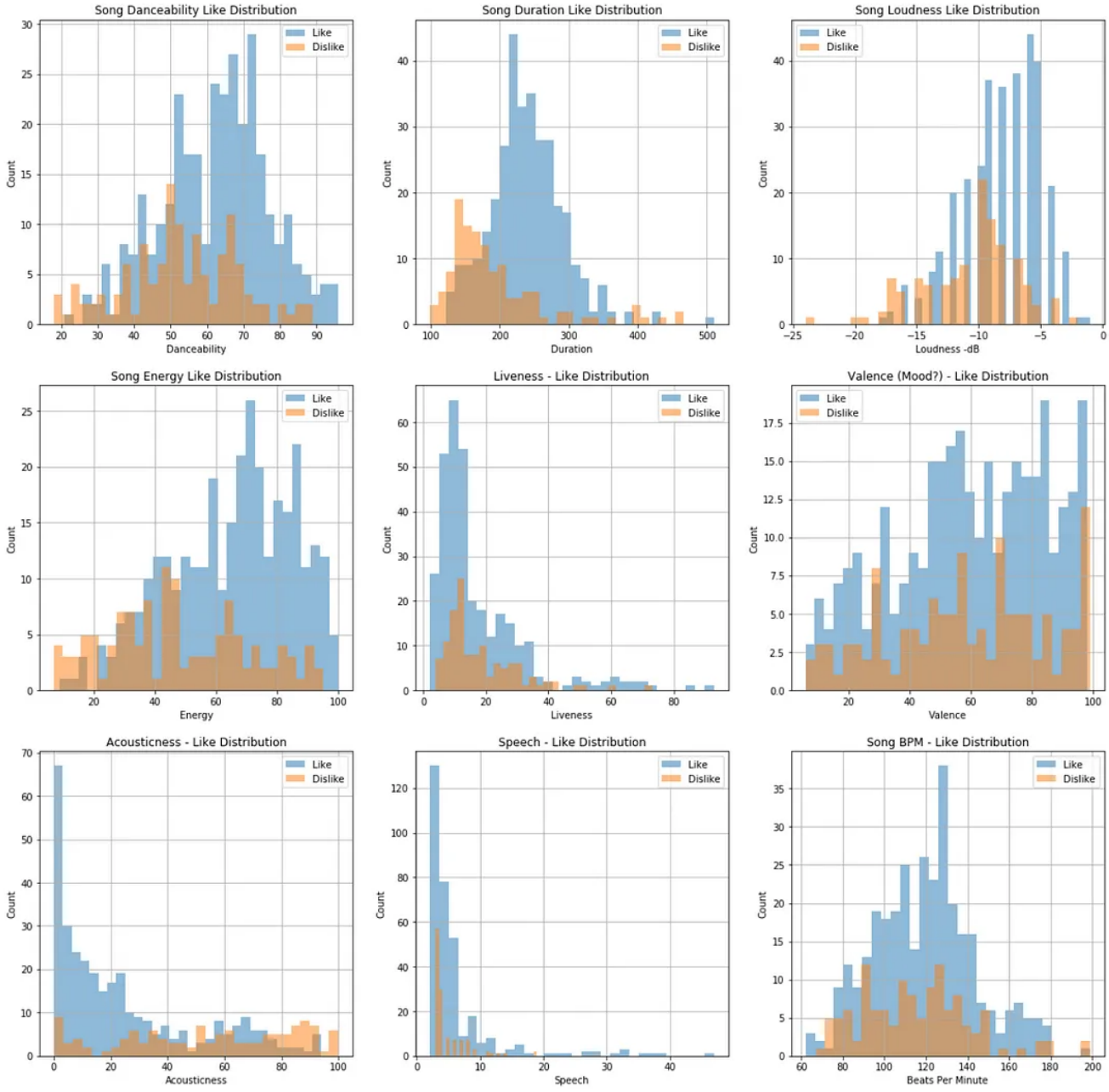


Image by Author

The distribution of songs to genre within the training set was then investigated. Within the training set, there were 86 unique genre names, which is far too many. Upon further investigation, many of these genre names were simply geographical differentials, for instance, “British rock” and “album rock” — which could be grouped within a “rock” category. Therefore the genres were generalized further, reducing the number of genres from 86 to 20.

```
class_train = class_train.replace({'top genre': {"album rock":
"rock", "glam rock": "rock", "dance rock":"rock", "art rock":"rock",
                                    "soft rock":"rock", "country
rock": "rock", "classic rock":"rock", "blues rock":"rock", "celtic
rock":"rock",
                                    "australian rock":"rock", "german
```

```
alternative rock":"rock", "alternative rock":"rock", "dance
pop":"pop",
                                    "brill building pop": "pop",
"europop": "pop", "barbadian pop": "pop", "classic uk pop":"pop",
                                    "new wave pop":"pop", "canadian
pop":"pop", "art pop":"pop", "belgian pop": "pop", "britpop": "pop",
                                    "italian pop":"pop", "classic danish
pop": "pop", "bow pop": "pop", "baroque pop": "pop", "bubblegum
pop": "pop",
                                    "afropop":"pop", "hip pop":"pop",
"atl hip hop": "hip hop", "east coast hip hop": "hip hop", "detroit
hip hop":"hip hop",
                                    "bronx hip hop": "hip hop",
"bubblegum dance": "dance", "eurodance":"dance", "belgian
dance":"dance", "german dance": "dance",
                                    "classic soul": "soul", "british
soul": "soul", "chicago soul": "soul", "british folk": "folk",
"american folk revival":"folk",
                                    "drone folk":"folk","canadian
folk":"folk", "deep adult standards":"adult standards", "glam
metal": "metal", "alternative metal": "metal",
                                    "acoustic blues":"blues", "british
blues":"blues", "louisiana blues":"blues", "g funk":"funk", "brit
funk":"funk",
                                    "afrobeat":"dance", "british
invasion":"rock", "doo-wop":"blues", "boy band":"pop",
"merseybeat":"rock-and-roll", "blue":"blues",
                                                    "bebop":"jazz",
"avant-garde jazz":"jazz", "boogaloo": "latin", "big room":
"trance", "bubble trance":"trance", "glam punk":"rock",
                                                    "australian talent
show":"pop", "mellow gold":"rock", "hi-nrg": "dance", "neo mellow":
"pop", "yodeling":"folk", "classic girl group":"pop",
                                                    "british dance
band":"jazz", "deep house":"dance", "uk garage": "dance", "chicago
rap":"hip hop"}})
```

The pie chart below shows the top 10 genres only to maintain a readable aesthetic. It is clear that the majority of the songs in the training dataset belong to the rock and pop category- the effect of this will be discussed in the conclusions.

```
# Find percent of each genre
df_genre = class_train['top genre'].value_counts()
[:10].sort_values(ascending=False) / len(class_train)
sizes = df_genre.values.tolist()
labels = df_genre.index.values.tolist()

# Pie chart for genre
fig1, ax1 = plt.subplots(figsize=(10,10))
ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=False,
textprops={'fontsize': 14})
```

```
ax1.axis('equal')
plt.title("Most Popular Genres\n" + "(limited to top 10 genres)",
bbox={'facecolor':'0.8', 'pad':5})
plt.show()
```

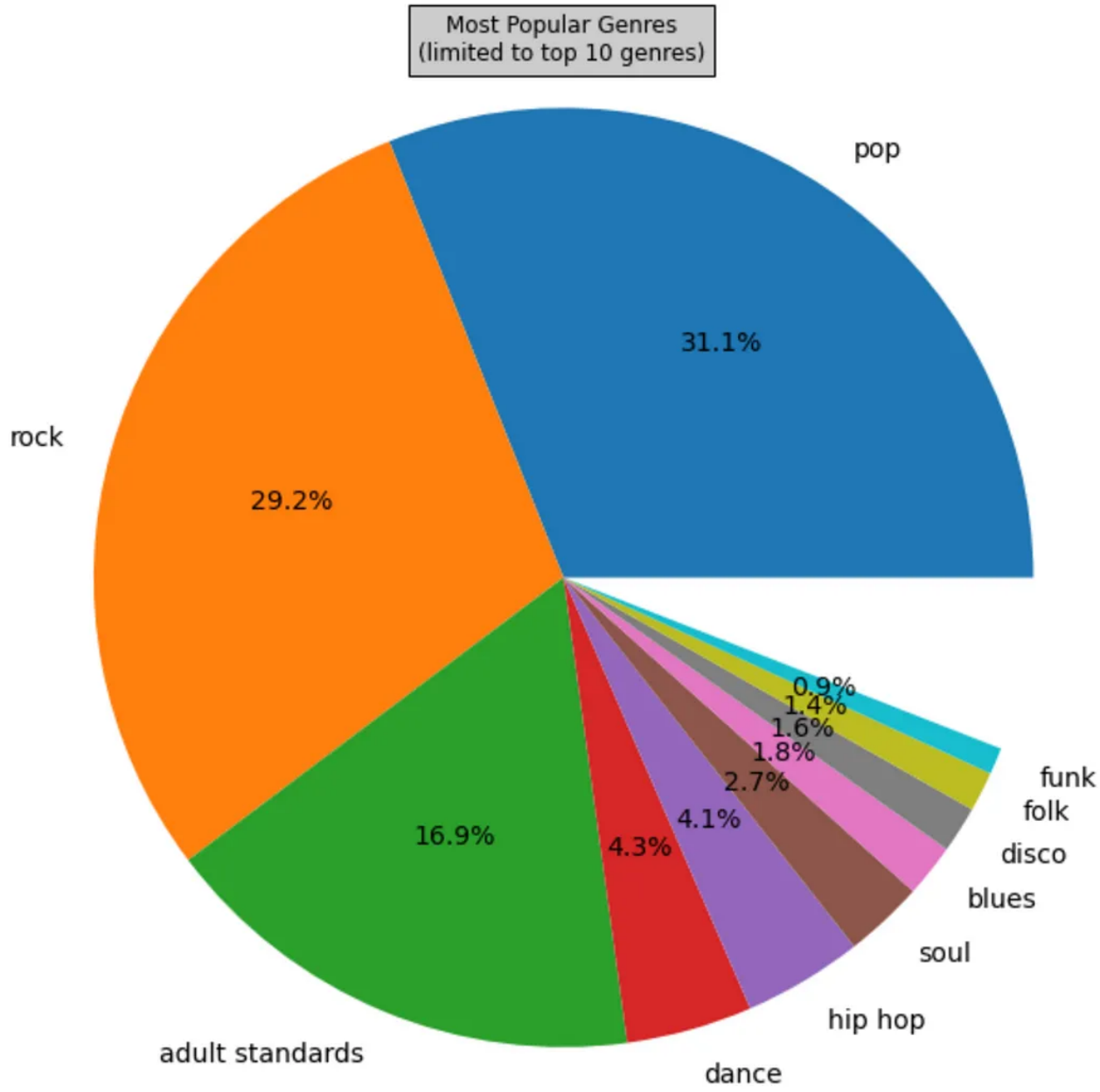


Image by Author

The distribution of the features was then looked at. The reason for this is we want each of the feature's histograms to display a normal distribution.

```
import matplotlib.pyplot as plt
%matplotlib inline
class_train.hist(bins=20, figsize=(15,15))
plt.show()
```

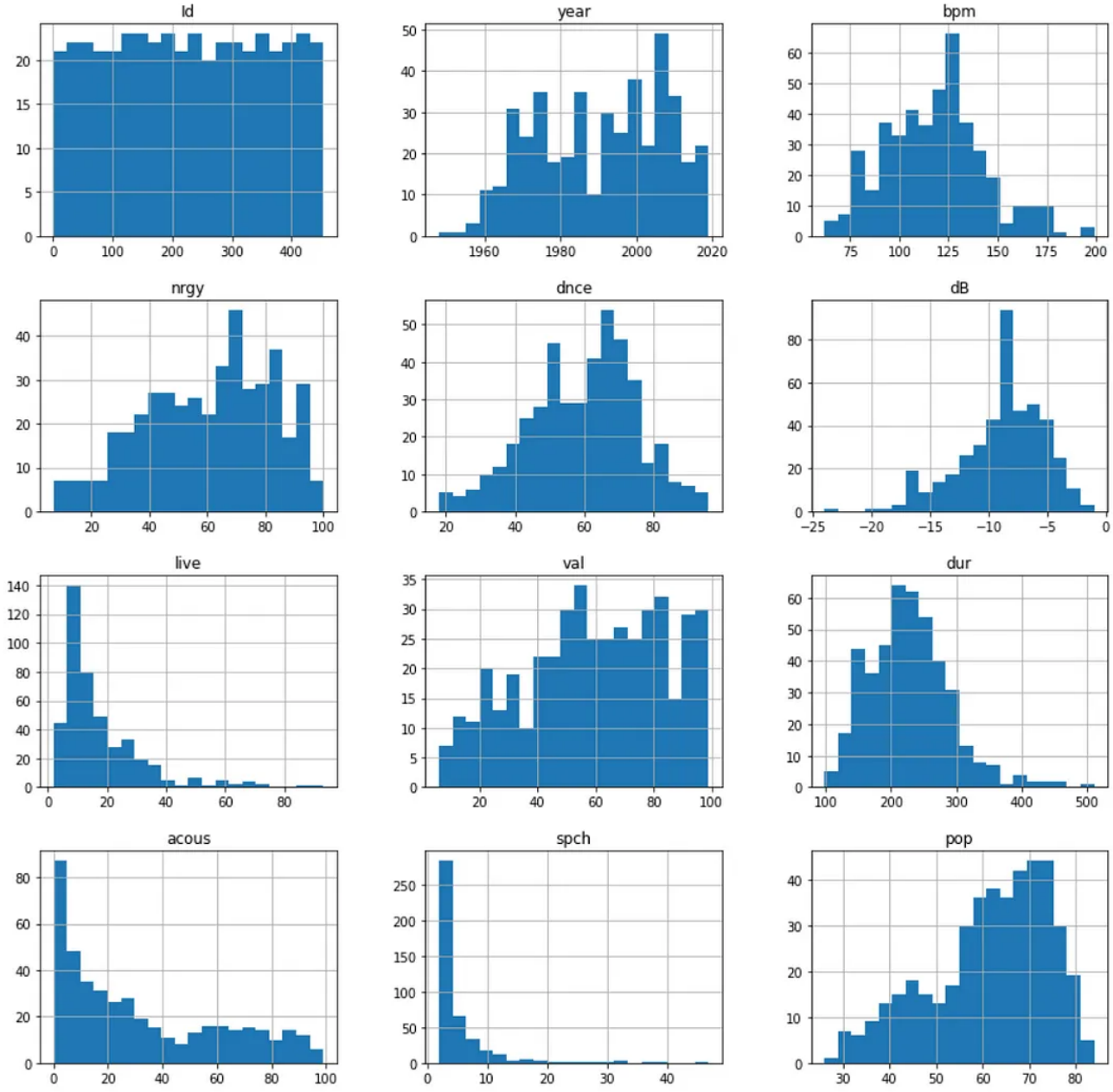


Image by Author

The histograms above show that some variables are skewed (acous, live, spch), are reason for this skew could be outliers. Therefore the skew could be remedied by removing potential outliers. Outliers can be identified using Box-plots, and a z-score. Data points with a z-score of more than 3 are removed. Removing outliers to reduce noise can improve a model's performance and prevent overfitting.

```
import seaborn as sns
```



```
sns.boxplot(x=class_train['live']) #OUTLIERS !!
```

```
# spch
sns.boxplot(x=class_train['spch']) #OUTLIERS !!

class_train = class_train[np.abs(class_train.dur-
class_train.dur.mean()) <= (3*class_train.dur.std())]
# keep only the ones that are within +3 to -3 standard deviations in
the column 'live'.

class_train = class_train[np.abs(class_train.live-
class_train.live.mean()) <= (3*class_train.live.std())]
# keep only the ones that are within +3 to -3 standard deviations in
the column 'live'.

class_train= df[np.abs(class_train.spch-class_train.spch.mean()) <=
(3*class_train.spch.std())]
# keep only the ones that are within +3 to -3 standard deviations in
the column 'spch'.
```

The features that will be used within the model creation process are columns 3 to 13. The features that were dropped are Title, Artist and Id. The features selected for the model training were: ‘year’, ‘bpm’, ‘nrgy’, ‘dnce’, ‘dB’, ‘spch’, ‘pop’, ‘live’, ‘acous’.

The training set was then separated using the train_test_split function to further partition the data into a 75% / 25% split. By further splitting the training dataset we have created a validation set.

```
y = class_train.values[:,14]
X = class_train.values[:,3:13]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)
```

Algorithm Selection + Model Performance + Improvement

These three sections are very much intertwined. As the algorithm selected is largely determined by how it performs in comparison to others, and in turn, improvements are very much an iterative process.

The Support Vector Machine (SVM) algorithm is a good choice for analysis as the training set has a small number of instances and a large number of features. This is suitable as the SVM algorithm can deal with high bias/low variance. The OneVersusRest is a heuristic method that involves splitting the training data into multiple binary classification problems. As previously noted this problem has been identified as multi-label. However the SVM algorithm requires that the positive

label has a numeric value of +1 and the negative label have a value of -1. Thus using the OneVersusRest technique a binary classifier it trained on each binary classification problem and predictions are made upon the most confident model.

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsRestClassifier

std_scaler = StandardScaler()
X_scaled_train = std_scaler.fit_transform(X_train)
X_scaled_train = std_scaler.fit_transform(X_train)

X_scaled_train = std_scaler.fit_transform(X_train)

svm_clf = OneVsRestClassifier(LinearSVC(C=1, loss = "hinge",
random_state = 1))

svm_clf.fit(X_scaled_train, y_train)
```

Scaling was carried out before building our SVM as they are sensitive to feature scales. This was to prevent the widest possible street in the model from being too close to the decision boundary.

Hyperparameters are not optimized by the learning algorithm and have to be done by us, the data analysts. Grid search is the most simple method for hyperparameter tuning, and it is available within scikit-learn Python machine learning library. On the first run of the model using the hyperparameter of C=1 the accuracy was very low at 26%. Grid search was used to reduce C (one of the algorithm's hyperparameters) to 0.01. This will prevent overfitting.

```
SVCpipe = Pipeline([('scale', StandardScaler()),
                    ('SVC',LinearSVC())])

# Gridsearch to determine the value of C
param_grid = {'SVC__C':np.arange(0.01,100,10)}
linearSVC =
GridSearchCV(SVCpipe,param_grid,cv=5,return_train_score=True)
linearSVC.fit(X_train,y_train)
print(linearSVC.best_params_)

svm_clf = OneVsRestClassifier(LinearSVC(C=0.01, loss = "hinge",
random_state=1))

preds = svm_clf.predict(X_scaled_train)
```

```
from sklearn.metrics import classification_report
print(classification_report(y_train,preds))
```

	precision	recall	f1-score	support
adult standards	0.51	0.44	0.47	55
blues	0.00	0.00	0.00	5
british comedy	0.00	0.00	0.00	1
chanson	0.00	0.00	0.00	2
dance	0.00	0.00	0.00	17
disco	0.00	0.00	0.00	6
disco house	0.00	0.00	0.00	1
folk	0.00	0.00	0.00	5
funk	0.00	0.00	0.00	2
hip hop	1.00	0.08	0.15	12
jazz	0.00	0.00	0.00	1
latin	0.00	0.00	0.00	2
metal	0.00	0.00	0.00	4
permanent wave	0.00	0.00	0.00	1
pop	0.50	0.73	0.60	105
r&b	0.00	0.00	0.00	1
rock	0.50	0.67	0.58	95
rock-and-roll	0.00	0.00	0.00	3
soul	0.00	0.00	0.00	7
trance	0.00	0.00	0.00	3
accuracy			0.51	328
macro avg	0.13	0.10	0.09	328
weighted avg	0.43	0.51	0.44	328

Image by Author

The final model accuracy that was achieved was 46%. Where accuracy is the number of correctly classified examples divided by the total number of classified examples.

The Logistic Regression algorithm, which is a classification algorithm, not a regression can be used for both binary and multi-class problems. The accuracy score obtained was 50%.

```
from sklearn.linear_model import LogisticRegression
ovr_clf = OneVsRestClassifier(LogisticRegression(max_iter=1000,
random_state=1))
```

```
ovr_clf.fit(X_train, y_train)
y_test_pred = ovr_clf.predict(X_test)

from sklearn.metrics import accuracy_score
confusion_matrix(y_test, y_test_pred)
print(accuracy_score(y_test, y_test_pred))
```

The Random Forest algorithm uses a modified tree learning algorithm to inspect a random subset of the features at each split in the "learning" process. This avoids the correlation of trees. Random Forest reduces the variance of the final model, reducing overfitting. Random Forest achieved an accuracy score of 46%.

```
from sklearn.ensemble import RandomForestClassifier
rnd_clf = RandomForestClassifier(n_estimators=25, max_leaf_nodes=16,
n_jobs=-1, random_state=1)
rnd_clf.fit(X_train, y_train)
ypred = rnd_clf.predict(X_test)
print(accuracy_score(y_test, ypred))
```

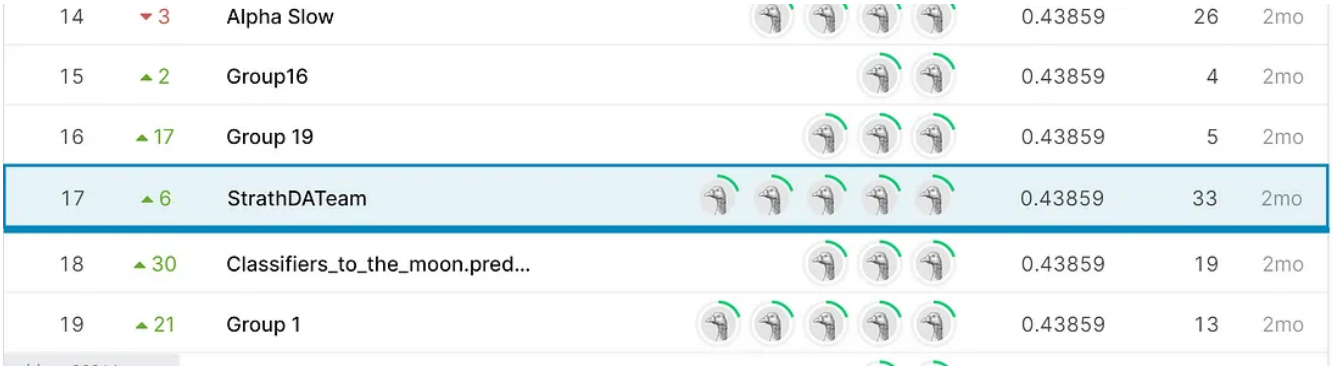
The best 3 models were selected to use for Ensemble learning: SVM, Logistic Regression and Random Forest. A hard voting classifier was used as it achieves high accuracy.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = OneVsRestClassifier(LogisticRegression(max_iter=1000,
penalty = "l2", C=1, random_state=1))
rnd_clf = RandomForestClassifier(random_state=1)
svm_clf = OneVsRestClassifier(LinearSVC(C=0.01, loss = "hinge",
random_state = 1))
voting_clf = VotingClassifier(estimators=[('lr', log_clf), ('rf',
rnd_clf), ('svc', svm_clf)],voting='hard')
voting_clf.fit(X_train, y_train)

from sklearn.metrics import accuracy_score
for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    ypred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, ypred))
```

The Logistic Regression and Voting Classifier had the same score. The Voting Classifier was selected as it is more robust as it reduces the spread of the predictions and model performance.

Conclusions

When submitted to the Kaggle Competition… an acceptable 17th place (out of 50 teams).



14	▼3	Alpha Slow	0.43859	26	2mo
15	▲2	Group16	0.43859	4	2mo
16	▲17	Group 19	0.43859	5	2mo
17	▲6	StrathDATeam	0.43859	33	2mo
18	▲30	Classifiers_to_the_moon.pred...	0.43859	19	2mo
19	▲21	Group 1	0.43859	13	2mo

The accuracy score was significantly lower when the selected model was applied to the test data (once submitted to the Kaggle competition). This is called overfitting, whereby the model where it predicts well the labels of the training data, but makes frequent errors when applied to new data. This means that the model has a high variance.

Possible solutions to Overfitting that were attempted:

- adding more Data
- removing outliers
- dimensionality reduction (via PCA)

By identifying the 15 null values and manually adding the genre labels within the training set it was attempted to increase the overall sample size. However, removing outliers using the z-score is then largely counterproductive. Overall the amount of data that was available to train the model was very limited and this is probably what led to overfitting.

```
# reload the data
class_train = pd.read_csv("CS98XClassificationTrain.csv")
class_test = pd.read_csv("CS98XClassificationTest.csv")
```

```
# update the genres for the missing values
class_train.loc[class_train['title'] == 'Unchained Melody', 'top
genre'] = 'pop'
class_train.loc[class_train['title'] == 'Someone Elses Roses', 'top
genre'] = 'adult standards'
class_train.loc[class_train['title'] == 'Drinks On The House', 'top
genre'] = 'pop'
class_train.loc[class_train['title'] == 'Pachuko Hop', 'top genre']
= 'blues'
class_train.loc[class_train['title'] == 'Little Things Means A Lot',
'top genre'] = 'blues'
class_train.loc[class_train['title'] == 'The Lady Is A Tramp', 'top
genre'] = 'pop'
class_train.loc[class_train['title'] == 'If I Give My Heart To You',
'top genre'] = 'pop'
class_train.loc[class_train['title'] == 'Happy Days And Lonely
Nights','top genre'] = 'rock'
class_train.loc[class_train['title'] == 'Stairway Of Love','top
genre'] = 'rock'
class_train.loc[class_train['title'] == 'You', 'top genre'] = 'pop'
class_train.loc[class_train['title'] == 'No Other Love' , 'top
genre'] = 'adult standards'
class_train.loc[class_train['title'] == "Hot Diggity" , 'top genre']
= 'folk'
class_train.loc[class_train['title'] == "Ain't That Just the Way" ,
'top genre'] = 'r&b'
class_train.loc[class_train['title'] == "I Promised Myself" , 'top
genre'] = 'pop'

# dropping NULL values  (I've Waited So Long Anthony Newley = ?
(Dance/ Electronic))
class_train = class_train.dropna(axis=0)

# check again for null values
class_train.info()
```

PCA (Principal Component Analysis) is a dimensionality reduction method. Dimensionality reduction removes redundant and highly correlated features in the dataset. It also helps reduce the overall noise in the data.

```
from sklearn.decomposition import PCA

pca = PCA()
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
print(pca.explained_variance_ratio_)
```

```
pca = PCA(n_components=6)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

Using PCA to reduce the dimensionality showed no improvement on the Kaggle Score. PCA is most effective when features have a strong correlation to each other. As highlighted earlier with the Correlation heatmap there are no strong correlations between features within the training dataset.

While the model did not perform brilliantly on the test data hopefully there were some useful lessons learned and visualizations that can be recast for your future projects!

An excellent free resource to learn more about ML is the Hundred Page Machine Learning Book by Andriy Burkov.

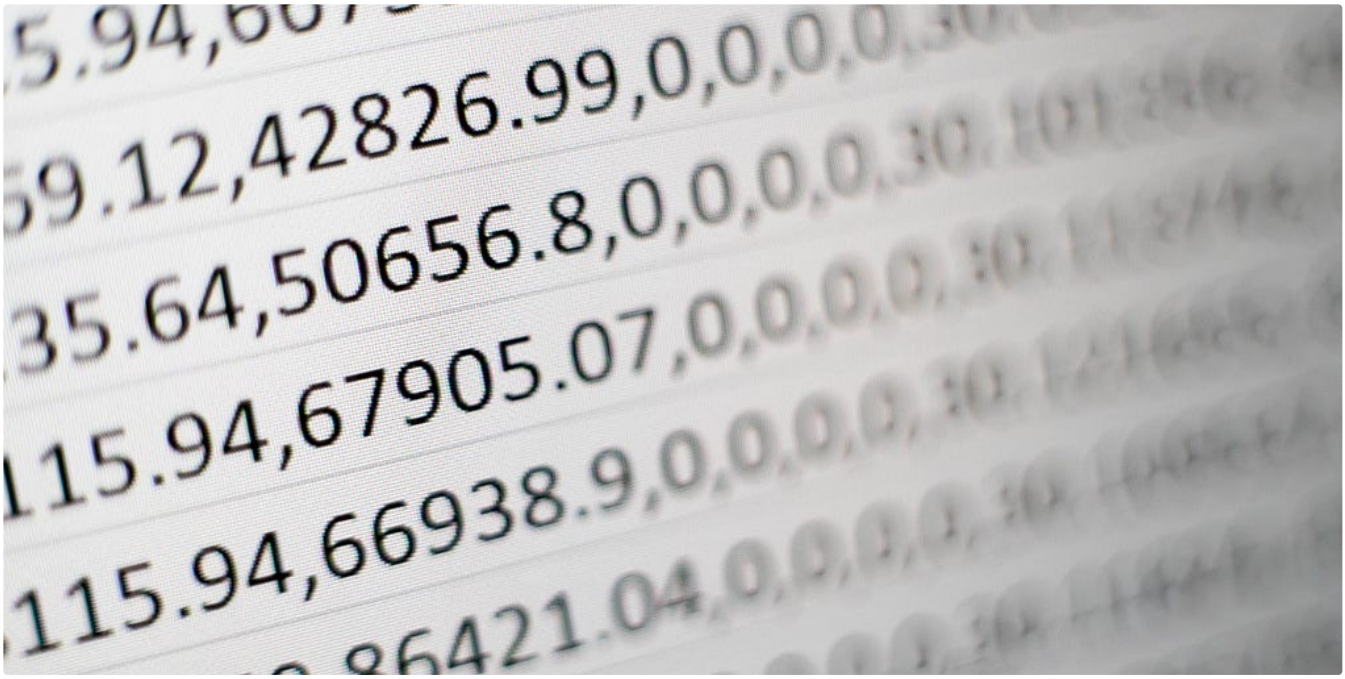
Spotify | Data Science | Classification | Machine Learning | Python

Follow

Written by Cd

83 Followers · Writer for Towards Data Science

More from Cd and Towards Data Science



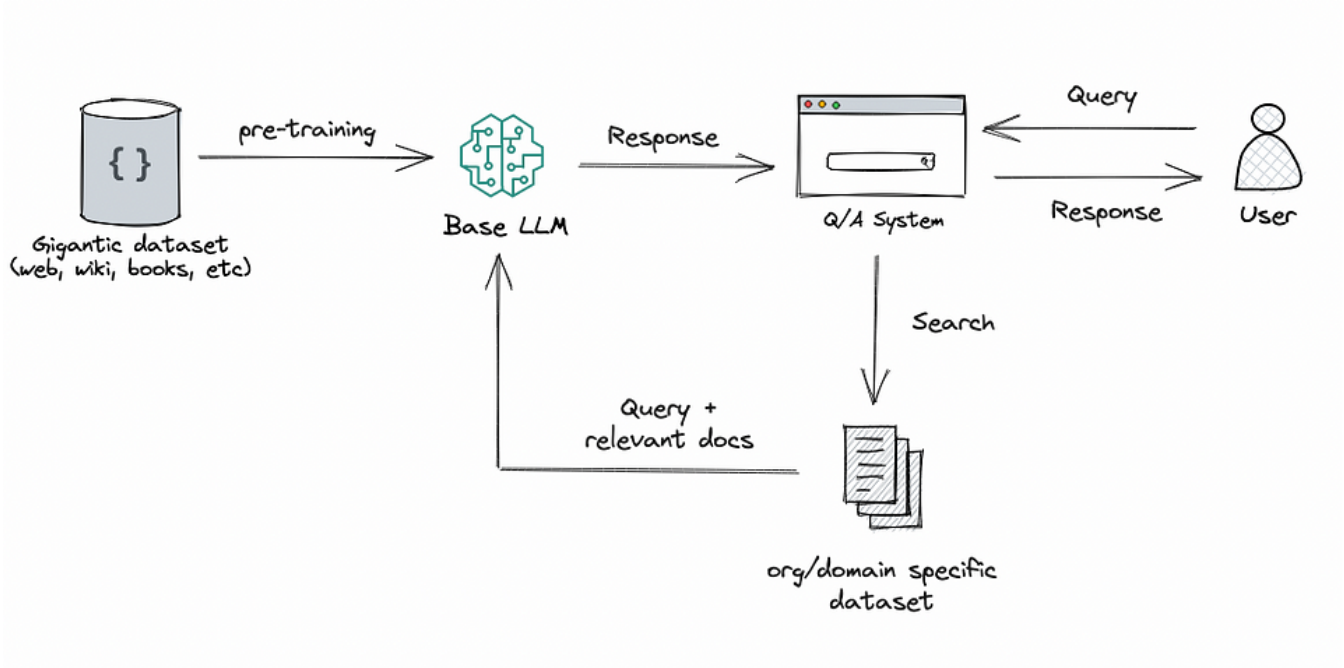
Cd in DataDrivenInvestor

Data Collection for 2000 Stocks using Python and APIs

A quick and concise guide to bulk concatenation of financial ratio data from FinancialModellingPrepAPI.

2 min read · Dec 24, 2021

42



Heiko Hotz in Towards Data Science

RAG vs Finetuning—Which Is the Best Tool to Boost Your LLM Application?

The definitive guide for choosing the right method for your use case

✨ · 19 min read · Aug 25

2K 16

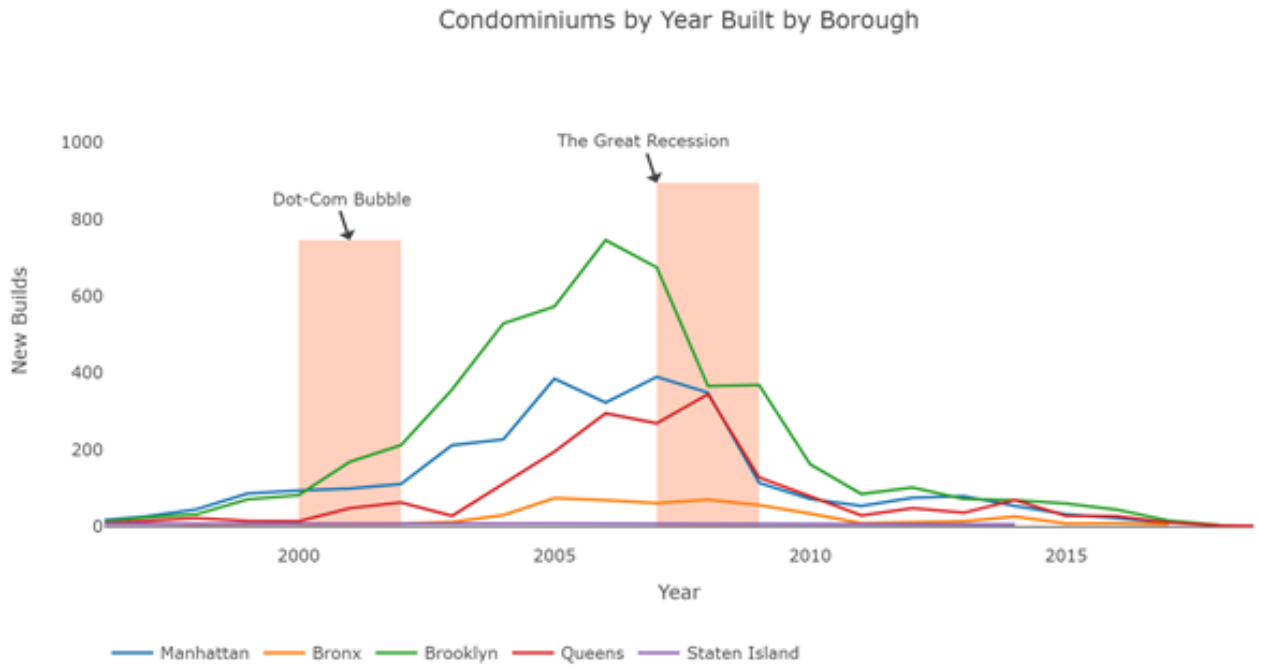
Giuseppe Scalamogna in Towards Data Science

New ChatGPT Prompt Engineering Technique: Program Simulation

A potentially novel technique for turning a ChatGPT prompt into a mini-app.

9 min read · Sep 4

1.2K 12



Cd in Towards Data Science

Data Analysis of New York City's Condo Market using Python

You can find my code on my github:

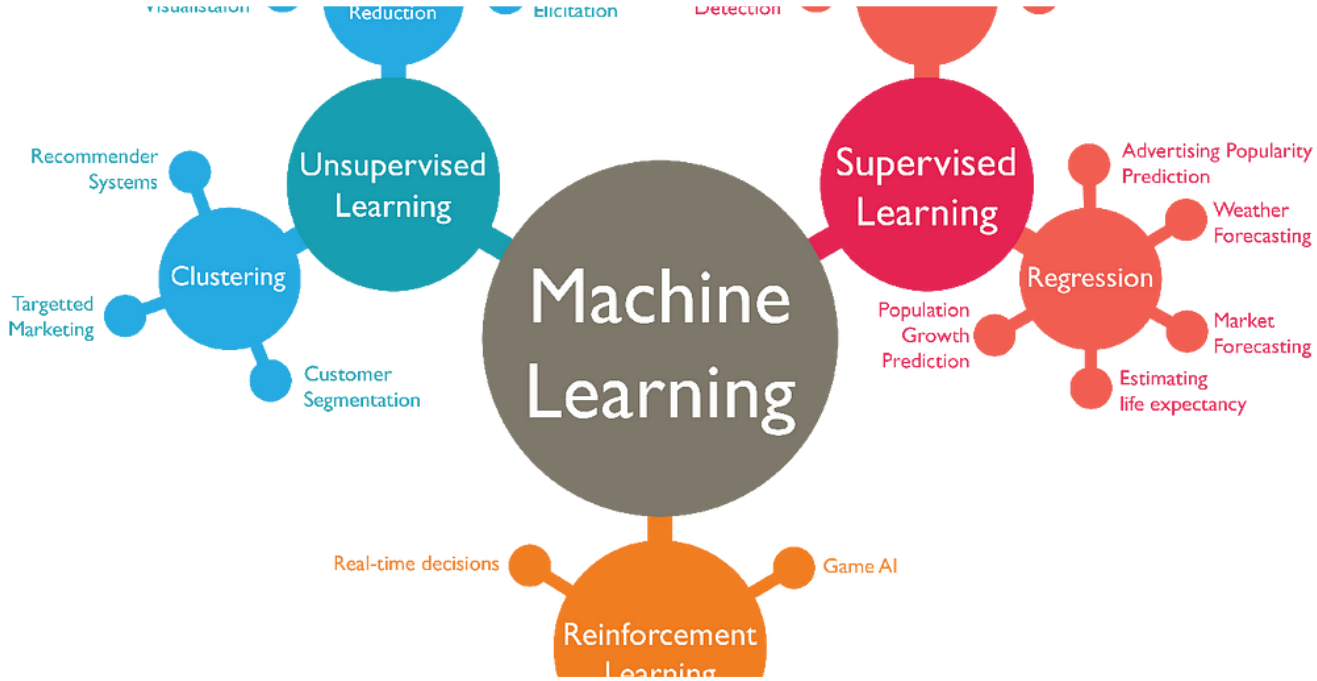
8 min read · Nov 18, 2020

73

See all from Cd

See all from Towards Data Science

Recommended from Medium



Nimra Shahzadi

Supervised Machine Learning: Classification and Regression

This article aims to provide an in-depth understanding of Supervised machine learning, one of the most widely used statistical techniques...

8 min read · May 29

2

Abbas Behrain

Creating an AI-powered Job Recommendation System

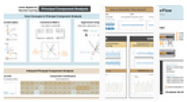
Job Recommendation System using Scraped Glassdoor Data, Machine Learning Techniques, and Streamlit Application

12 min read · Jun 13

20

Lists

Predictive Modeling w/ Python
20 stories · 418 saves



Practical Guides to Machine Learning
10 stories · 481 saves

Coding & Development
11 stories · 194 saves

New_Reading_List
174 stories · 117 saves

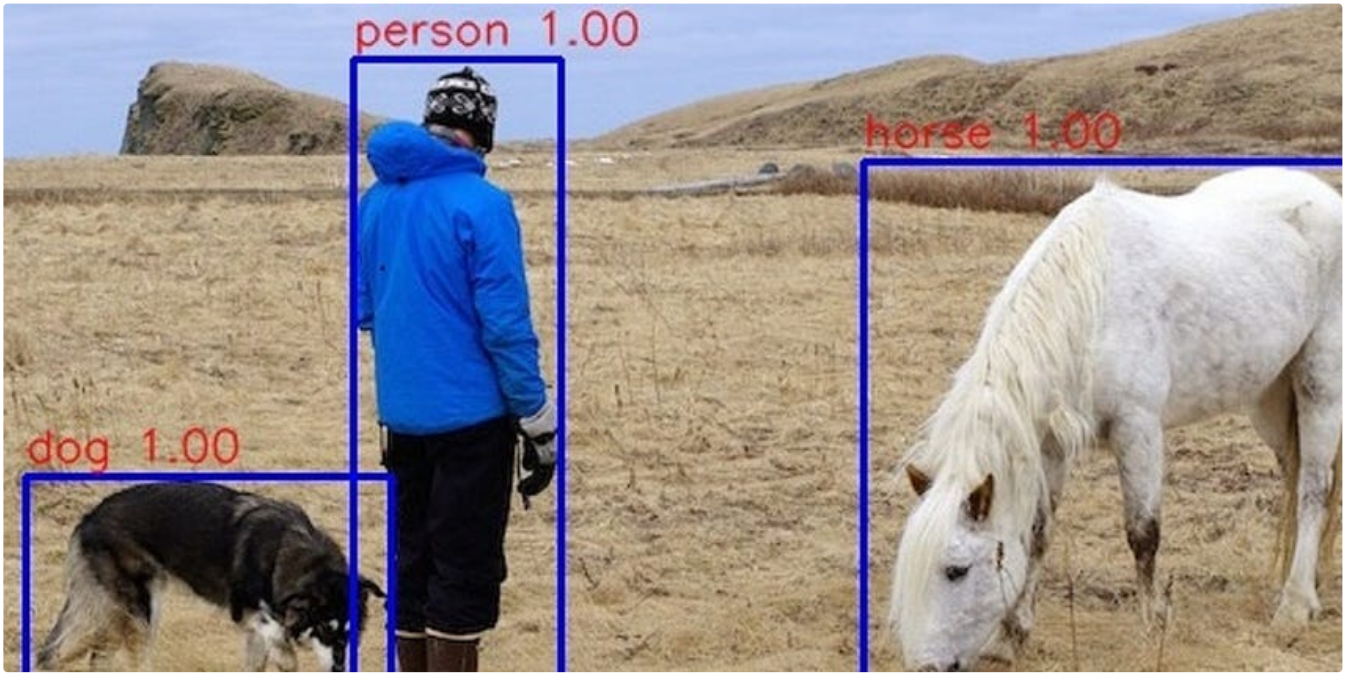
Zach Quinn in Pipeline: Your Data Engineering Resource

3 Data Science Projects That Got Me 12 Interviews. And 1 That Got Me in Trouble.

3 work samples that got my foot in the door, and 1 that almost got me tossed out.

· 7 min read · Aug 30, 2022

3.5K 48



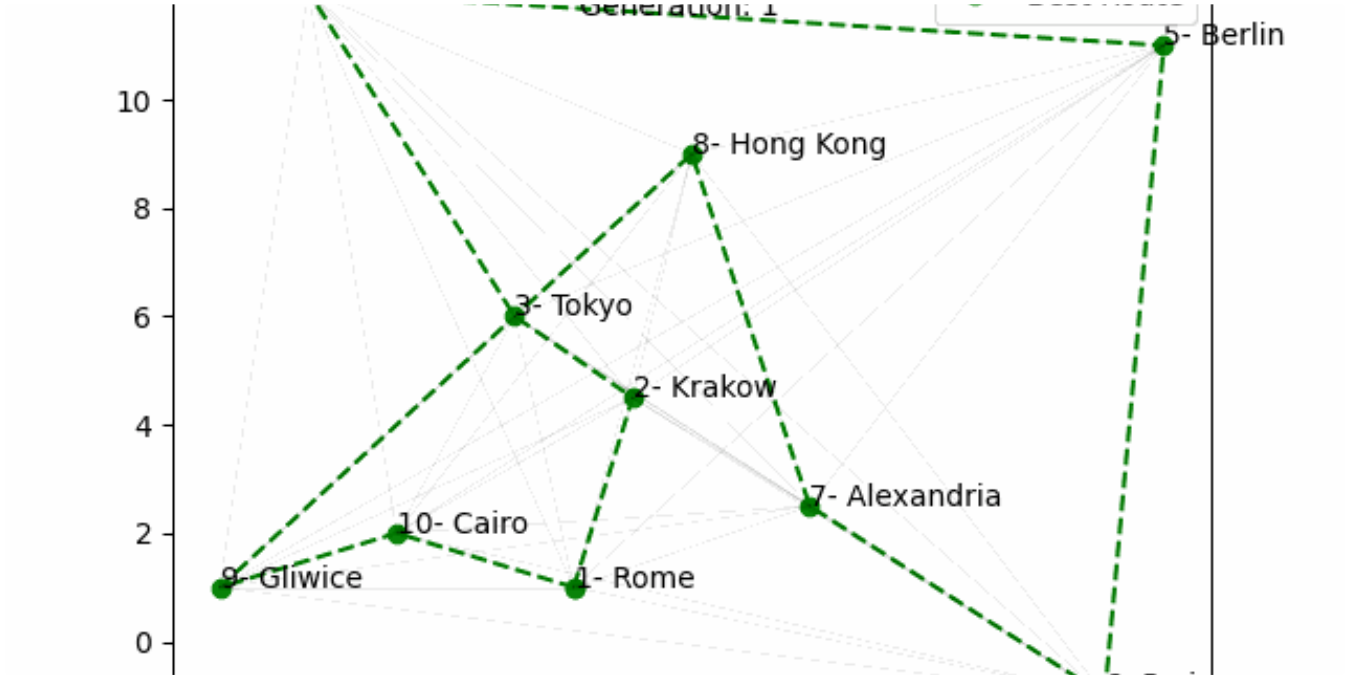
Abdul4code

Introduction to Object Detection Algorithms

Object detection is a remarkable computer vision Technique which enables the computer to identify specific objects within an image and to...

12 min read · Aug 4

23

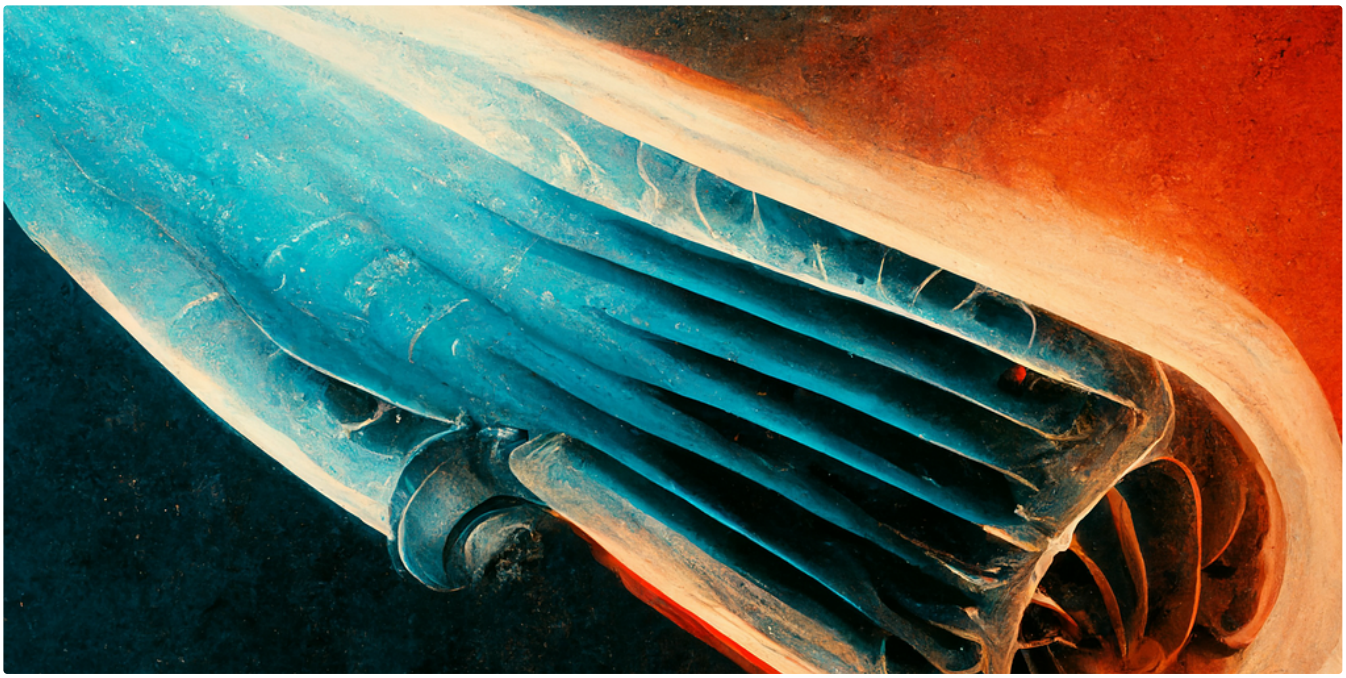


Ramez Shendy in AI monks.io

Traveling Salesman Problem (TSP) using Genetic Algorithm

Traveling Salesman Problem

· 15 min read · Aug 6



Jacob Ferus

Predictive Maintenance in Python — Exploratory Analysis and Visualization

Exploratory analysis of the CMAPSS Simulated Jet Engine Dataset in Python with Pandas and Plotly.

· 9 min read · Oct 6, 2022

134 1