Water Potability Prediction

CS542 Final Project

Team2: Keshav Maheshwari
Weiqi Ji
Renjian Zheng
Yiqin Zhang

Water - the most significant resource of life

Crucial for supporting the life of most existing creatures and human beings.

Drinking Water Crisis

- 1.1 billion people lack access to an improved drinking water supply;
- 1.8 million people die from diarrheal disease each year.
- Improved water supply and sanitation, and better management of water resources, can boost countries’ economic growth and can contribute greatly to poverty reduction.

Objective

- Explore the different features related to water potability
- Create a model to determine whether or not the sample tested from the water body is fit for human consumption.
- Binary Classification
- Predict water potability.

What has already been done for this dataset ?

Task Submissions

Water Quality | EDA | LuciferML | 73% accuracy
lucif3r · 4 days ago · Notebook · Python · 42 Comments

Exploring and Predicting Drinking Water Potability
Thomas Konstantin · an hour ago · Notebook · Python · 20 Comments

Potable Water 🌊 ~71% acc.
FunkyMonk · 12 days ago · Notebook · Python · 23 Comments

Water Potability Prediction - Acc 79%, F1 - 0.7
Subhankar Bhattacharya · 11 days ago · Notebook · Python · 7 Comments

water_potability_test_acc68
Serhanayberkkilic · a month ago · Notebook · Python · 1 Comment

💧 Water Quality Analysis and Ensemble Modeling
Rohan Lone · 8 days ago · Notebook · Python · 8 Comments

Quality_of_Drinking_Water 💧
Parvez Sohail · 10 days ago · Notebook · Python · 3 Comments

[Water Quality]Plotly+Optuna+LGBMClassifier
Ranjeet shrivastav · 2 days ago · Notebook · Python · 5 Comments

Prediction_Precision70_(Beginner_Friendly)
Nakshatra Jagtap · a month ago · Notebook · Python · 0 Comments

Water portability prediction with 66% accuracy.
GurmanjotSingh Cheema · 2 months ago · Notebook · Python · 0 Comments

Water Quality (potability) Prediction
Nilshan Sadaruwan · 18 days ago · Notebook · Python · 0 Comments

Dataset

Raw Dataset

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0
...	...	...	...	...	...	...	...	...	...	...
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821	1
3272	7.808856	193.553212	17329.802160	8.061362	NaN	392.449580	19.903225	NaN	2.798243	1
3273	9.419510	175.762646	33155.578218	7.350233	NaN	432.044783	11.039070	69.845400	3.298875	1
3274	5.126763	230.603758	11983.869376	6.303357	NaN	402.883113	11.168946	77.488213	4.708658	1
3275	7.874671	195.102299	17404.177061	7.509306	NaN	327.459760	16.140368	78.698446	2.309149	1

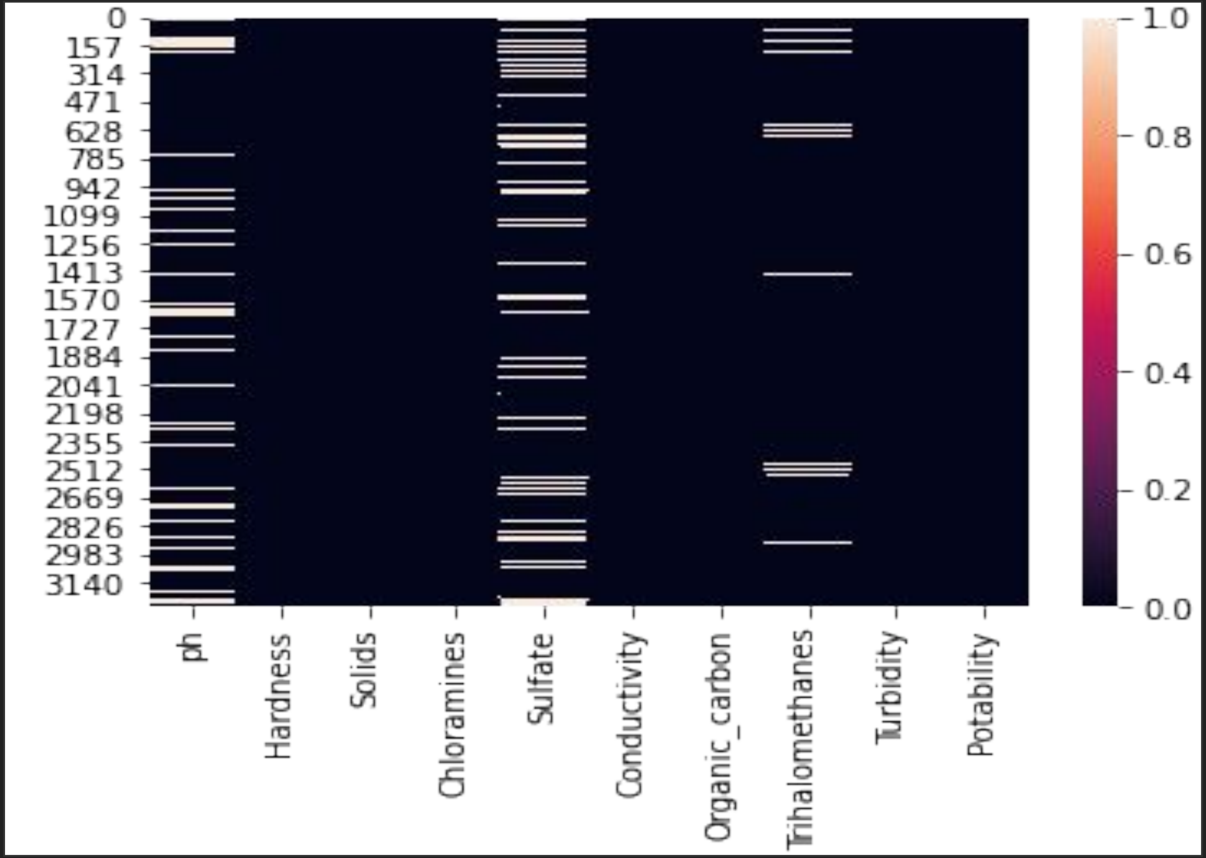
3276 rows × 10 columns

Clean Data

Split the dataset into training dataset and testing dataset

Normalize data

Impute the missing values with the class conditional mean



NaN density Map

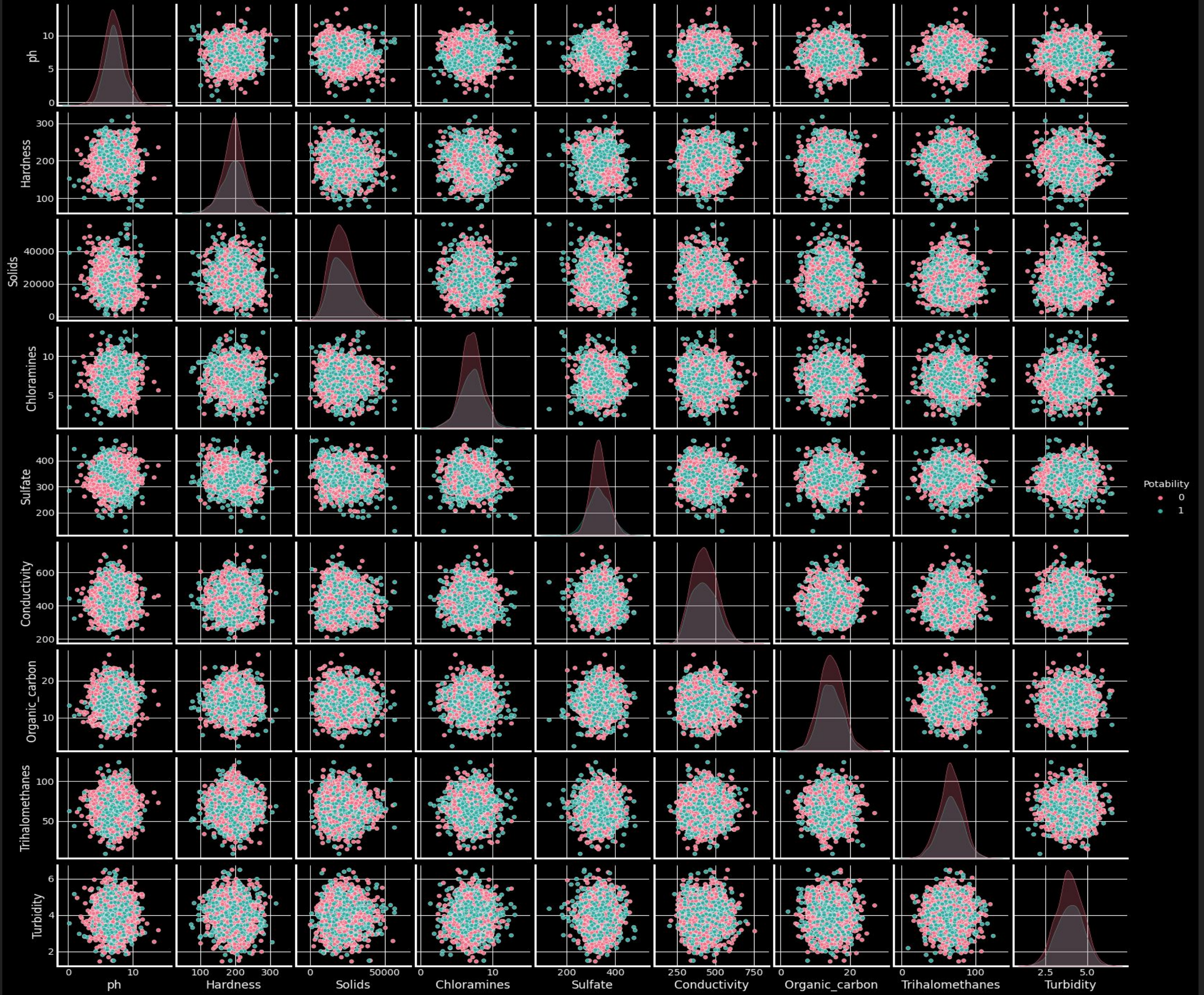
Processed+Normalized Dataset

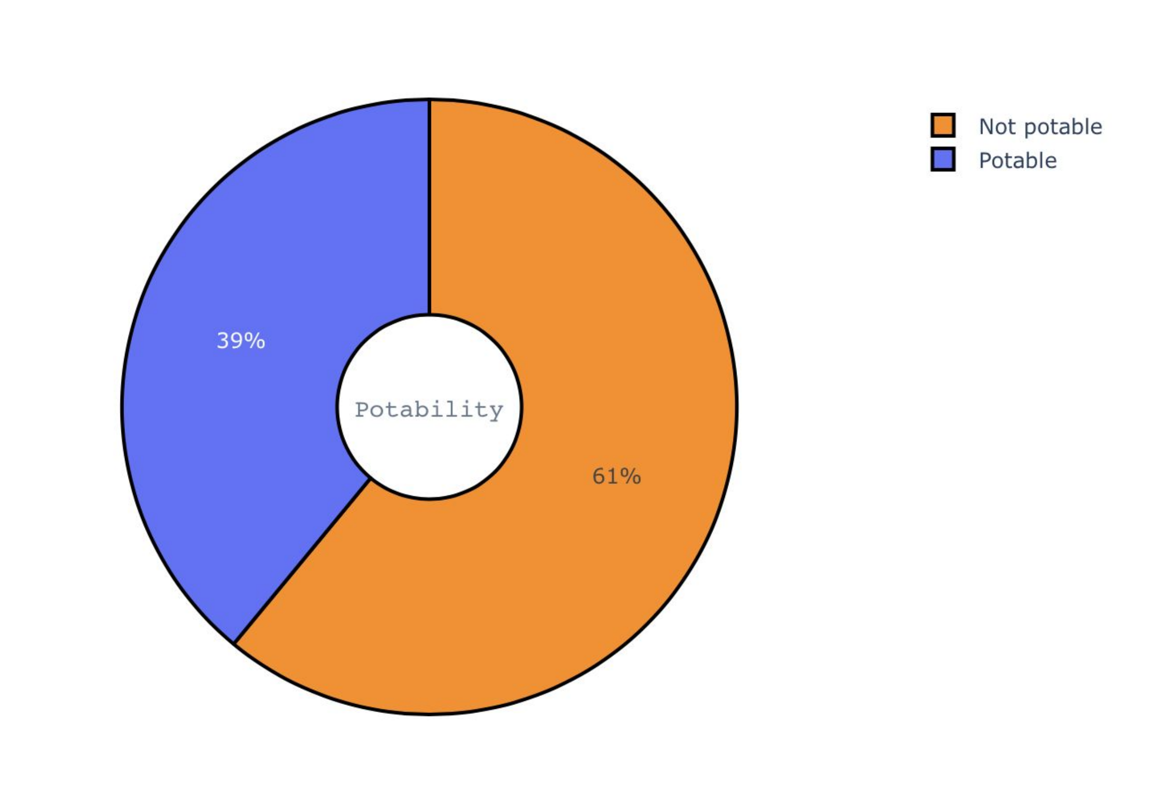
	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	0.674652	0.356824	0.210940	0.711739	0.515817	0.719015	0.245456	0.623383	0.458530	1.0
1	0.644632	0.292590	0.320800	0.599955	0.486180	0.471549	0.481642	0.592559	0.479344	1.0
2	0.505270	0.444492	0.378987	0.639353	0.578265	0.514270	0.410885	0.406645	0.245285	1.0
3	0.485723	0.705773	0.637413	0.716219	0.165243	0.340946	0.353734	0.592553	0.454706	1.0
4	0.512438	0.565765	0.329690	0.573762	0.449068	0.234439	0.472548	0.597651	0.470769	1.0
...	...	...	...	...	...	...	...	...	...	...
3271	0.472624	0.535616	0.557526	0.569913	0.583939	0.412657	0.430366	0.382989	0.410735	0.0
3272	0.552469	0.665553	0.352273	0.513236	0.583939	0.373573	0.491707	0.746048	0.448375	0.0
3273	0.497970	0.502281	0.453797	0.537183	0.627059	0.406715	0.677506	0.287526	0.335857	0.0
3274	0.336370	0.477740	0.360726	0.502722	0.790891	0.331867	0.612521	0.696002	0.455679	0.0
3275	0.373572	0.468936	0.453992	0.567200	0.809344	0.452684	0.468338	0.584928	0.301685	0.0

3276 rows × 10 columns

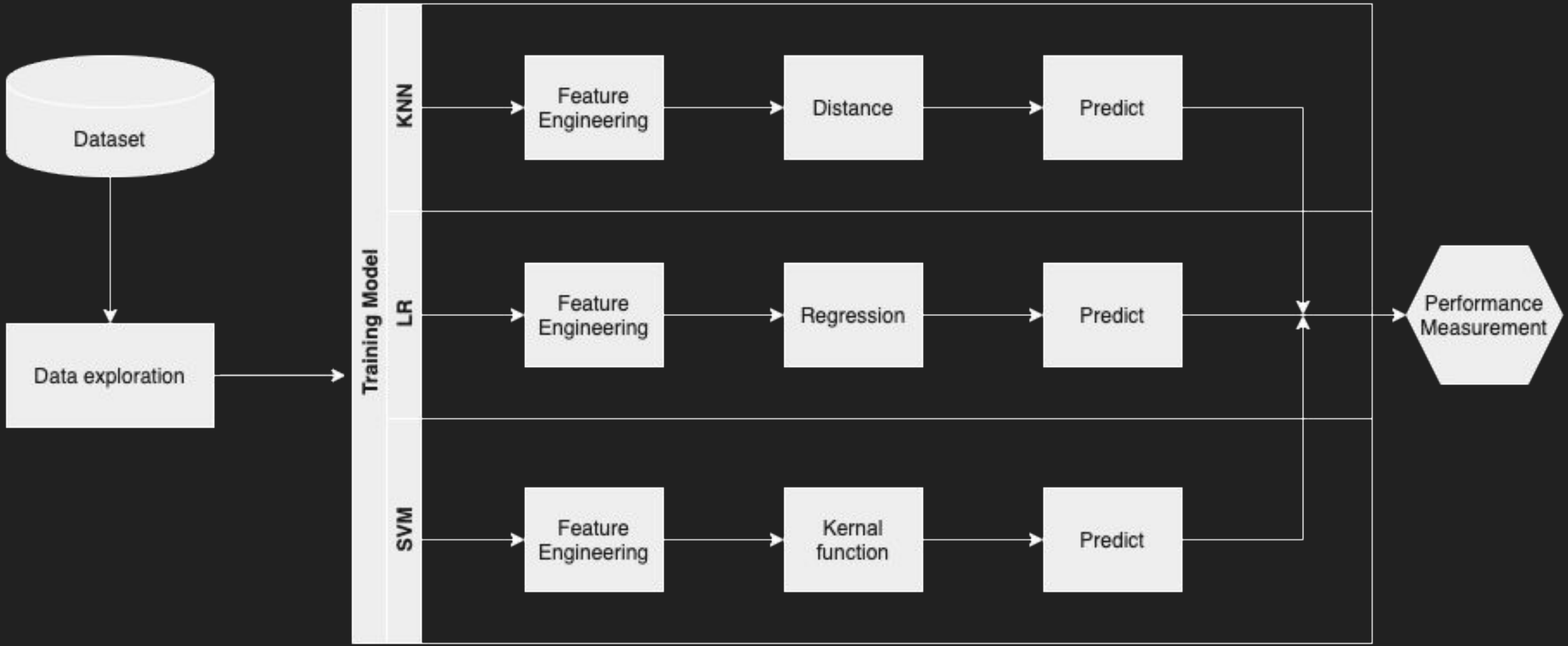
Data Distribution

- ph
- Hardness
- Solids
- Chloramines
- Sulfate
- Conductivity
- Organic_carbon
- Trihalomethanes
- Turbidity




Not potable
Potable
39%
Potability
61%

Work Flow


Dataset
Data exploration
Training Model
KNN
LR
SVM
Feature Engineering
Distance
Predict
Feature Engineering
Regression
Predict
Feature Engineering
Kernal function
Predict
Performance Measurement

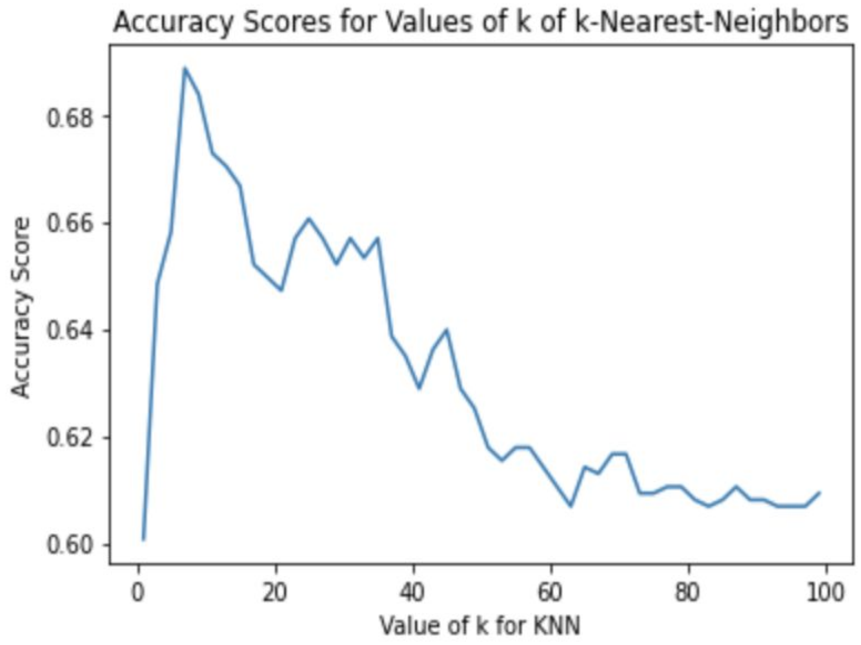
Model Training

KNN

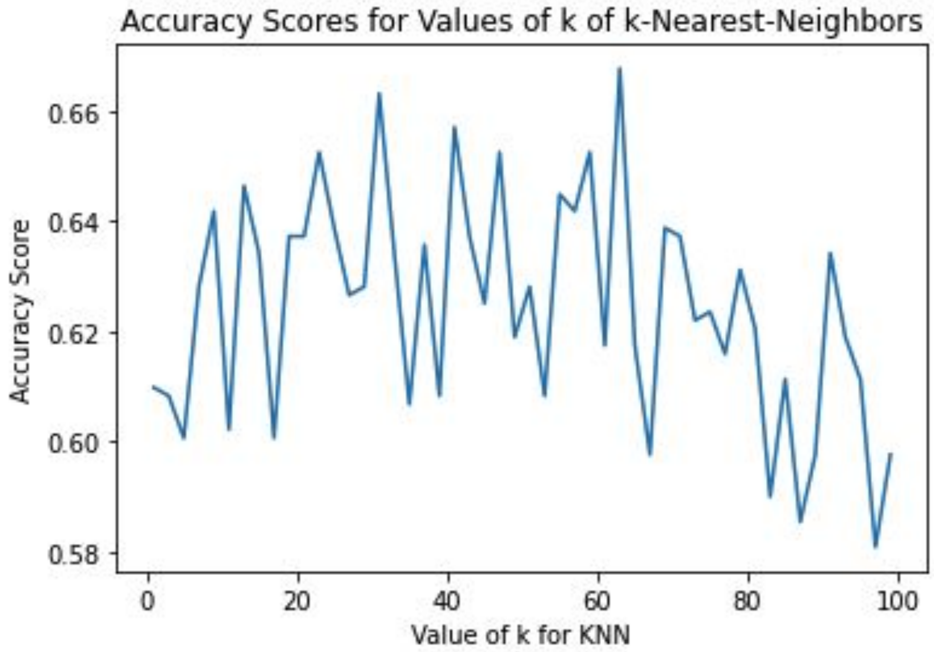
1. Euclidean Distance Metric
2. Sort to get closest N neighbors
3. Get the most frequent classification among these N neighbors
4. Return the predicted classification.

K=9, Accuracy=0.69

K=63, Accuracy=0.67



Our Model

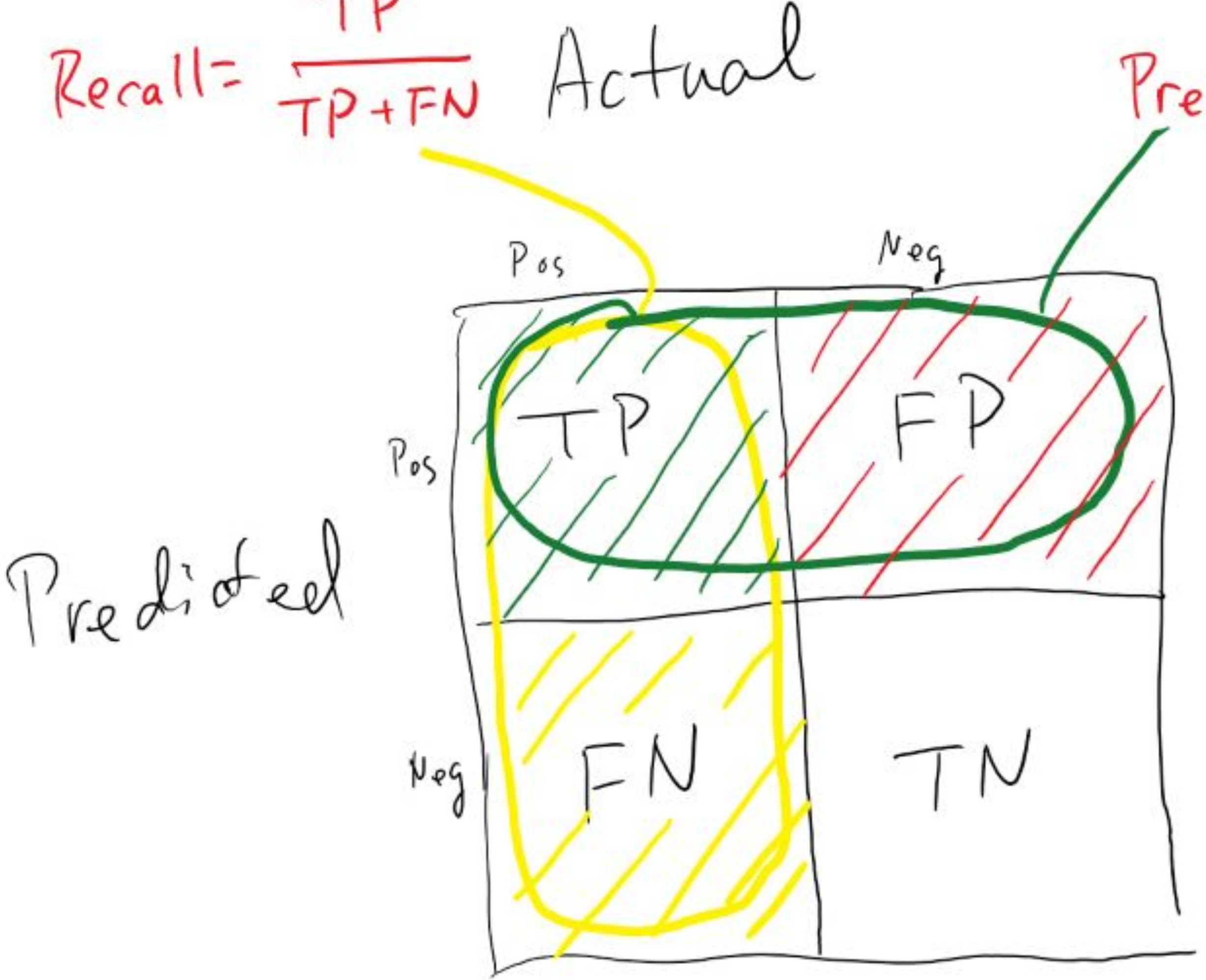


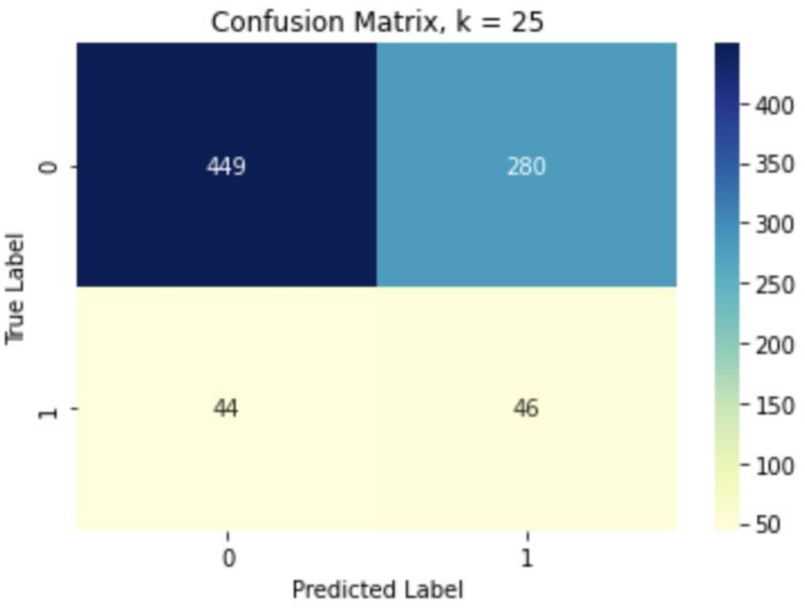
Sklearn Model

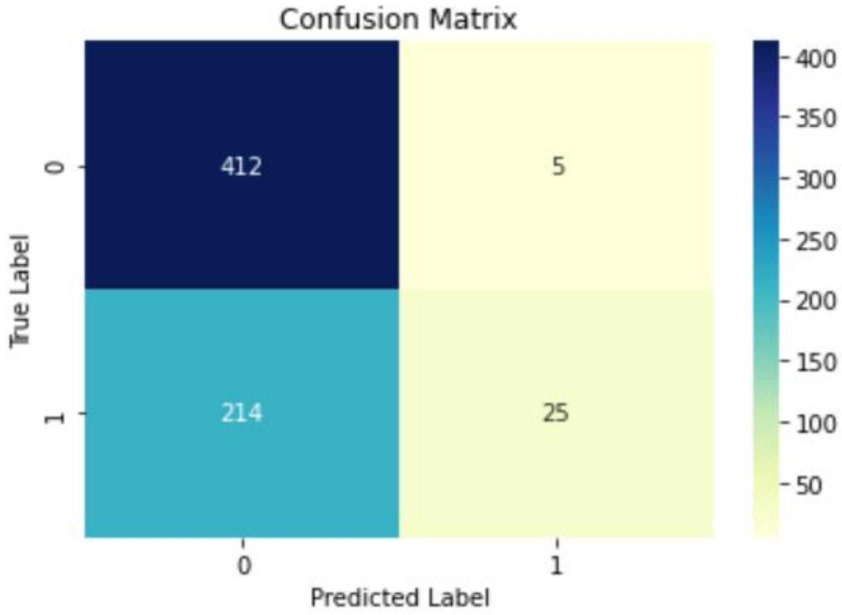
$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Accuracy} = \frac{TP + TN}{\text{Total}}$$







	precision	recall	f1-score	support
0.0	0.91	0.62	0.73	729
1.0	0.14	0.51	0.22	90
accuracy			0.60	819
macro avg	0.53	0.56	0.48	819
weighted avg	0.83	0.60	0.68	819

```
########### Now k is 63 ###############
----- Evaluation on Test Data -----
Accuracy Score:  0.6676829268292683
```

0.67

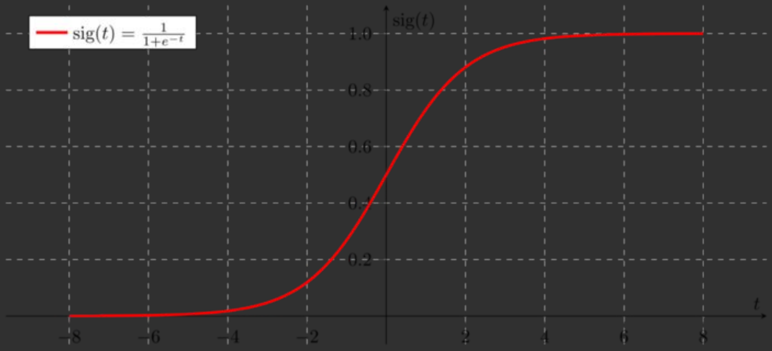
	precision	recall	f1-score	support
0	0.66	0.98	0.79	418
1	0.79	0.11	0.20	238
accuracy			0.67	656
macro avg	0.73	0.55	0.49	656
weighted avg	0.71	0.67	0.58	656

Observation

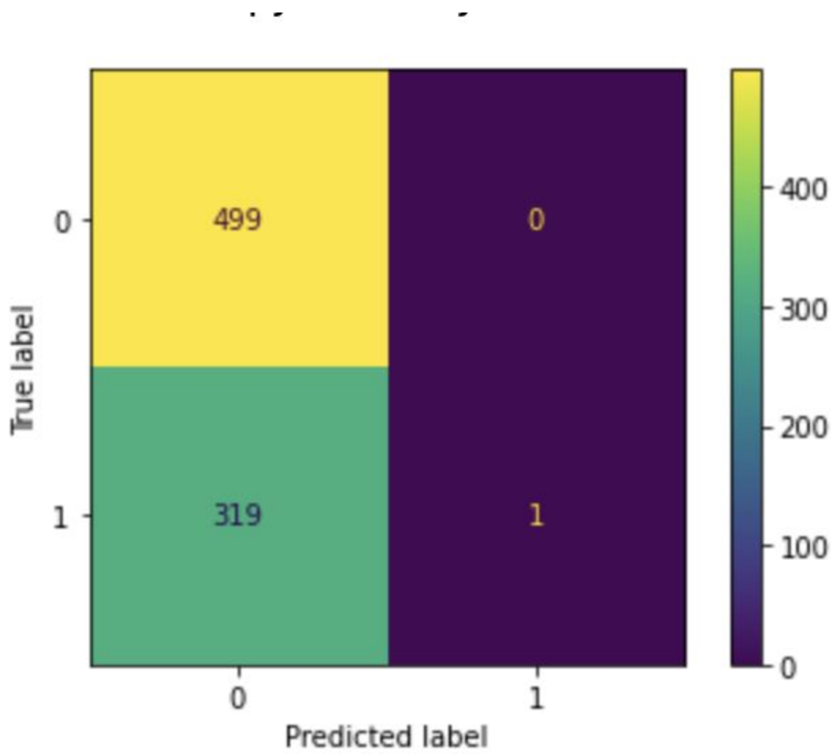
Advantages	Disadvantages	Solution
The algorithm is easy to implement. **Simplicity**, no need to tune several parameters, or make additional assumptions. **Reliability**, as KNN is produced the highest accuracy for us.	**Slow** and computationally intensive. To produce a graph of up to 100 neighbors, it was predicted to take 12 hours until completion.	However, **parallel** programming and SCC's 32 core computers saved the day in 6 minutes.

Logistic Regression

1. Linear Assumption.
2. Sigmoid Function



0.61



	precision	recall	f1-score	support
0	0.61	1.00	0.76	499
1	1.00	0.00	0.01	320
accuracy			0.61	819
macro avg	0.81	0.50	0.38	819
weighted avg	0.76	0.61	0.46	819

Observations

1. Great at predicting 0’s, rarely predicts 1’s

2. Baseline accuracy as good as the percentage of 0’s in the datasets

3. There is a bug we haven’t worked out
    a. perhaps we should change the sigmoid function, becaust it’s not predicting 1’s

SVM

1. Stochastic gradient descent (SGD)
2. Kernal function
    a. Polynomial Kernel Function
    b. Radial Basis Function
3. Tune Parameters
    a. Gamma matrix
    b. C

SVC

Cost function:

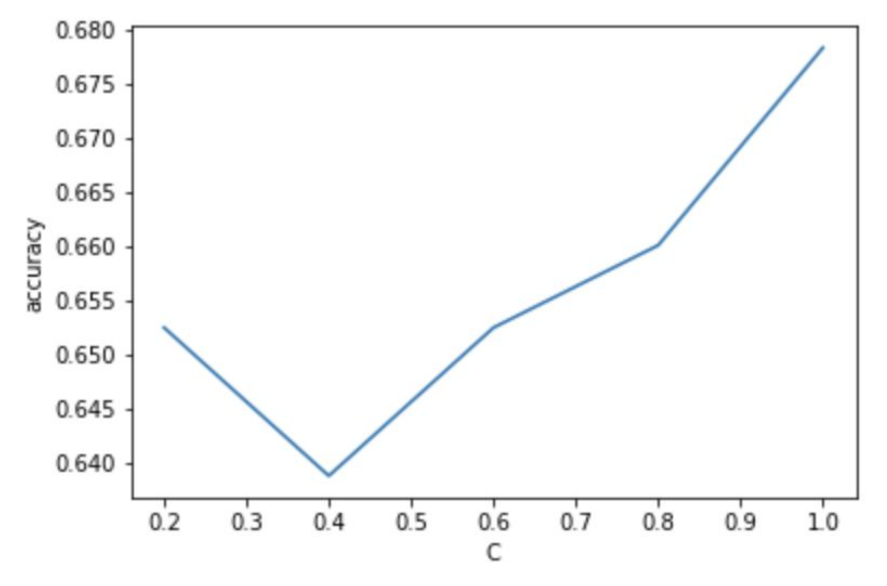
$$J(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2 + C\left[\frac{1}{N}\sum_{i}^{n} \max(0,\ 1 - y_i * (\mathbf{w} \cdot x_i + b))\right]$$

The gradient of the cost function:

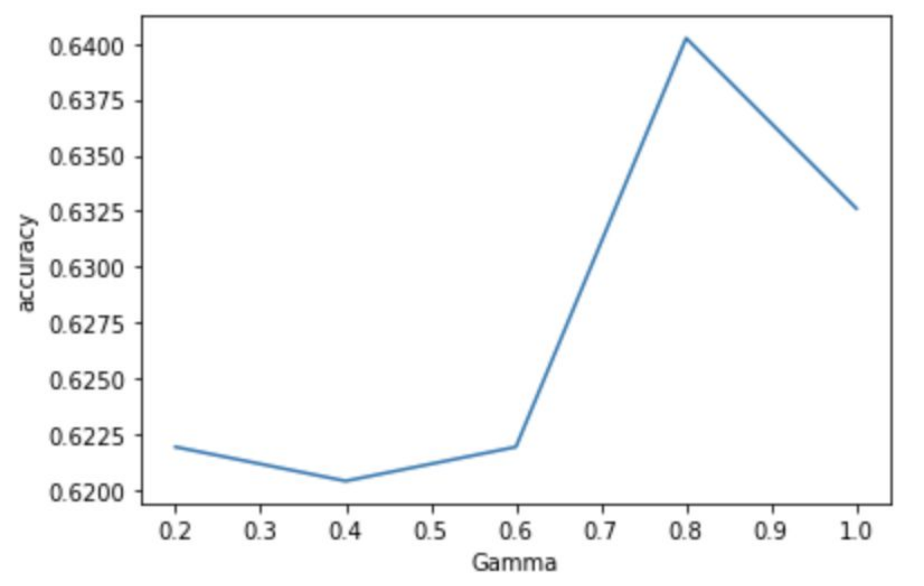
$$\nabla_w J(\mathbf{w}) = \frac{1}{N}\sum_{i}^{n} \begin{cases} \mathbf{w} & \text{if } \max(0,\ 1 - y_i * (\mathbf{w} \cdot x_i)) = 0 \\ \mathbf{w} - C y_i\, x_i & \text{otherwise} \end{cases}$$

SVM with Polynomial Kernel Function

Tune for C values. Here we use C = {0.2, 0.4, 0.6, 0.8, 1.0}. Gamma stays as 1.0



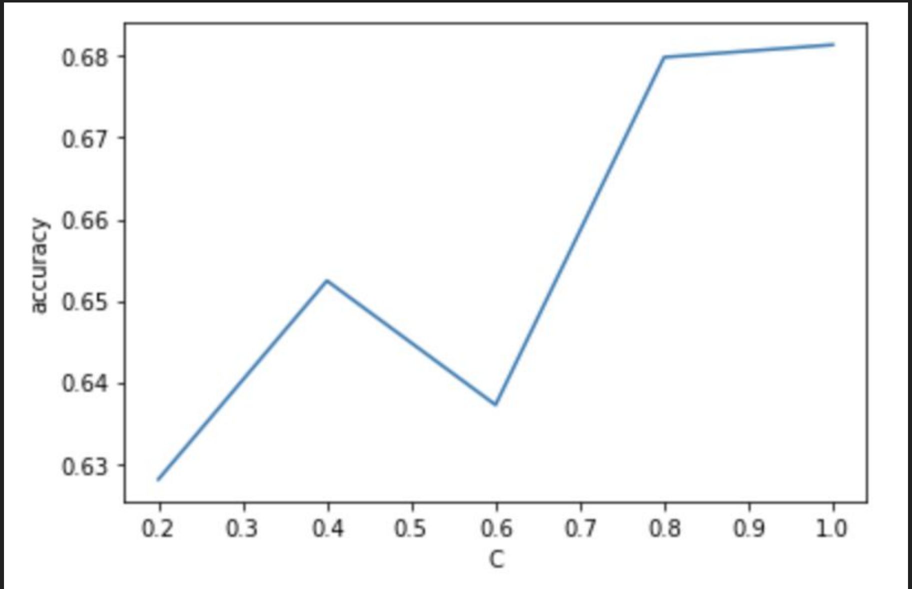
Tune for gamma. Here we use gamma = {0.2, 0.4, 0.6, 0.8, 1.0}. C stays as 1.0

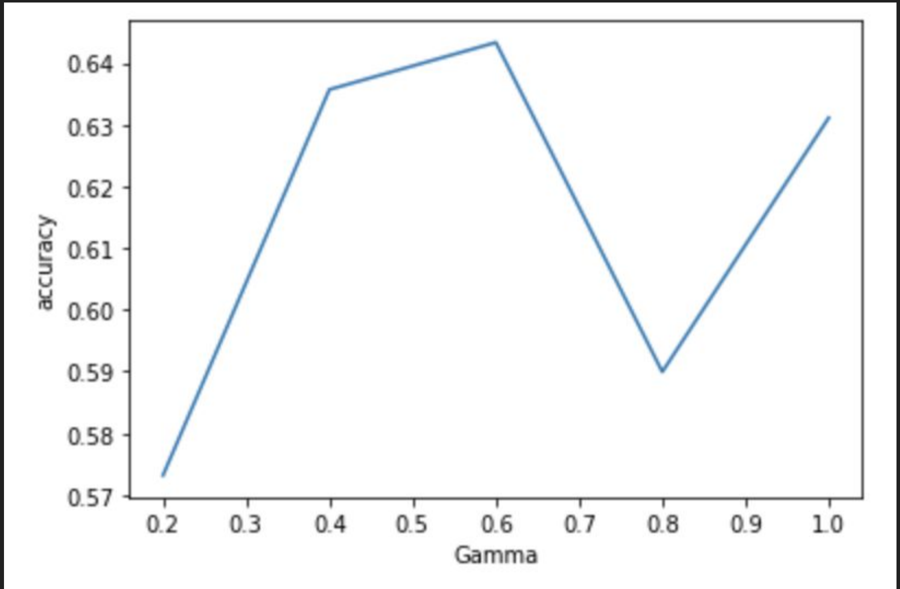


SVM with Radial Basis Function

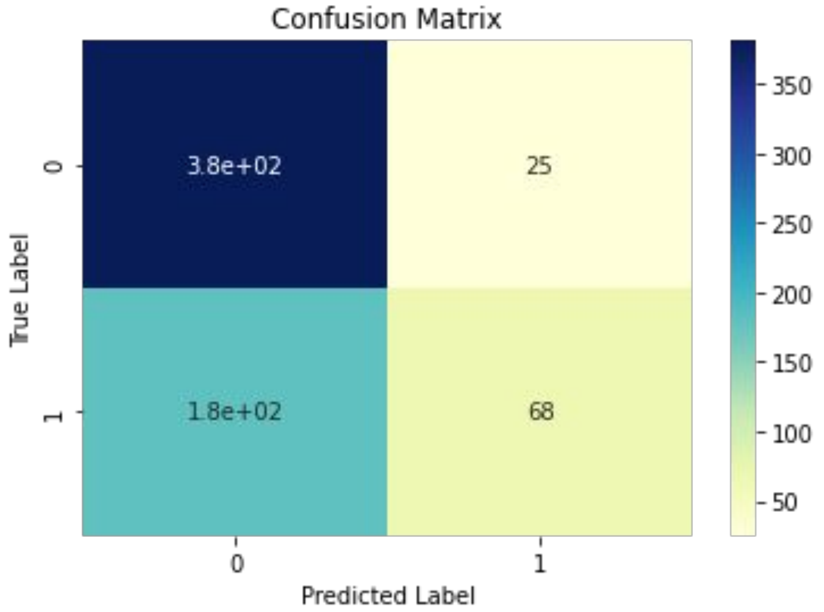
Tune for C values. Here we use C = {0.2, 0.4, 0.6, 0.8, 1.0}. Gamma stays as 1.0

Tune for gamma. Here we use gamma = {0.2, 0.4, 0.6, 0.8, 1.0}. C stays as 1.0





0.70




```
----- Evaluation on Test Data -----
Accuracy Score:  0.7027439024390244
              precision    recall  f1-score   support

        -1.0       0.70      0.94      0.80       415
         1.0       0.73      0.30      0.42       241

    accuracy                           0.70       656
   macro avg       0.72      0.62      0.61       656
weighted avg       0.71      0.70      0.66       656

-----------------------------------------------------
```

Observations

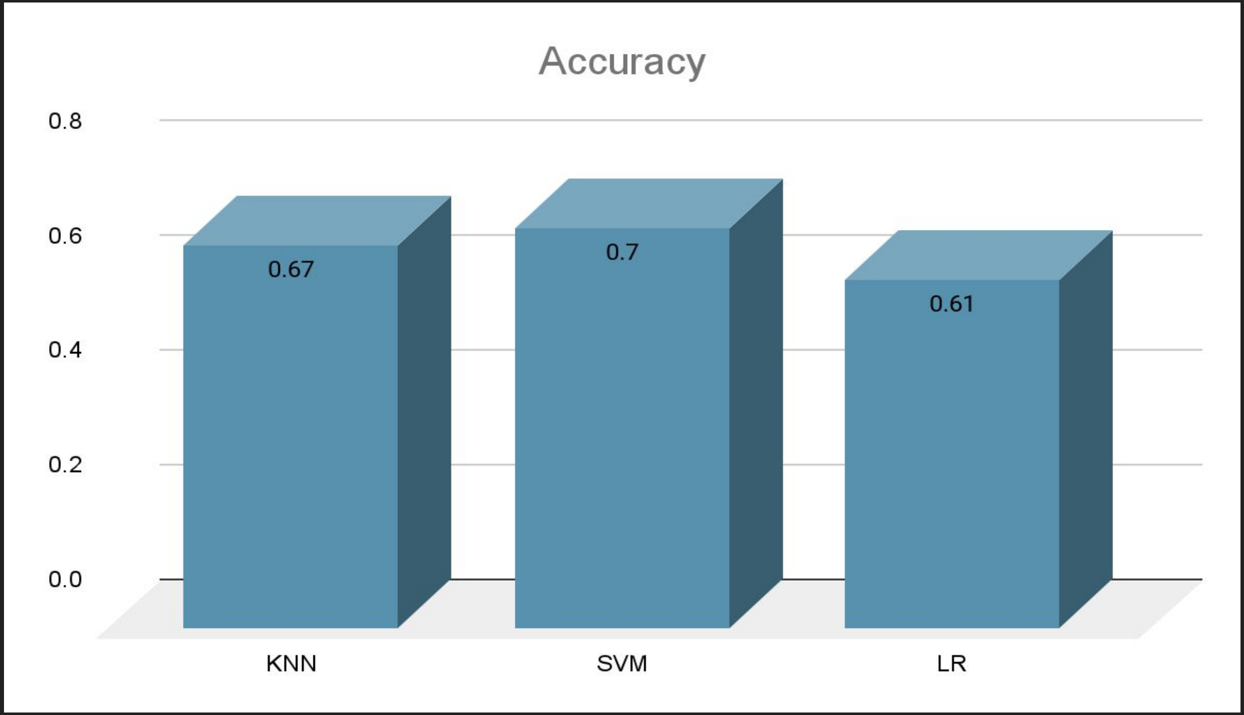
1. The SVM model has several unique benefits for solving datasets that are NOT linearly separable.

2. Different from KNN, there are a couple of parameters that can be tuned, i.e. C, gamma, and different kernel functions to choose from.

3. SVC does not work well with nonlinearly separable dataset. When we implemented our own SVC model (without kernel function) on the dataset, we got undesirable results, which is the reason why we switch to SKLearn.

Conclusion

SVM performs best on this classification appropriate dataset.



Conclusion

- Modeling and prediction of water quality are very important for the protection of the environment. Developing a model by using advanced artificial intelligence algorithms can be used to measure the future water quality and potentially save millions of lives in developing countries.

- The SVM algorithm implemented through SKLearn has achieved the highest prediction accuracy for the Water Potability dataset as compared with KNN and Logistic Regression algorithms.

Thank you!


People with no idea about AI, telling me my AI will destroy the world
Me wondering why my neural network is classifying a cat as a dog..
Dog