**Build and Evaluate a Machine Learning Model to solve a real-world  
problem with an appropriate dataset.**

Ryan T. Needham, 19156892

Computer and Data Science, CMP6202

Dr Hossein Ghomeshi

Birmingham City University

Abstract

How can we effectively build and evaluate a Machine Learning Model? Can an algorithm be used to solve real world problems? The premise of this report was to tackle how and why machine learning models are useful. Throughout the project a machine learning model was built, which used classification algorithms on a real-world dataset (Poisonous Mushroom Dataset). Several python packages were used in the process to help visualize and interpret the results of the data. It can be said that the results shown helped towards showing the viability of machine learning models as all classifications models portrayed great success in solving a targeted problem. The report highlighted how models need to go through a comparative process that evaluates all possible classifications algorithms in order to identify which model had the highest accuracy in solving real-world problems. It should be noted that the project only uses one dataset so, if the project was to be re-done, the optimal way to get better results is to compare the outcomes of this dataset with others.

Introduction

Machine Learning in its simplest form is a method that allows for data-scientists to make an automated prediction or assumption on a dataset that has been fed into a machine. Machine Learning is often classified into Supervised Learning and Unsupervised Learning. For the context of the report, it should be noted that the problem we aim to tackle will be through Supervised Learning means. The key objective of this report is building and optimizing a Machine Learning model’s accuracy given a dataset, whilst seeing if it can help solve real-world problems. When assessing and creating the machine learning model we will be using 4 key packages: ‘Pandas’, ‘Matplotlib’, ‘NumPy’ and ‘Sklearn’. The dataset that will be used is the ‘Poisonous Mushroom’ dataset. The mushroom dataset will require a classification (Supervised Learning) approach due to its discrete number values; so, to successfully document the Machine Learning’s model predictive power, the report will also be a comparative study on different methods of classifications, and which one works best.

The reason we want to include a comparative objective that looks at different classification methods is that the report has more academic weight to it and can be viewed as significant in

its findings. It can be noted that the report also looks to highlight how certain un-

common methods of classification can still be beneficial, and how

often popular methods of classification are not needed. The report itself can be deemed

important or a significant study due to its relativity to modern times.

The hypothesis or assumption at the ending of the report is that not one classification

method will stand out above the rest, saying that the only true aim of this report is to see if the Machine Learning problem can accurately detect a real-world problem, and in this case, it’ll be the detection of ‘poisonous mushrooms’

The Dataset

The ‘Poisonous Mushroom’ Dataset itself has 23 features(columns) and 8124 observations(rows). By using the ‘pandas’ python package we can use some minimal python code to extract how many null values are in our dataset

**print(**df**.**shape**)**

**print(**df**.**head**)**

**print(**df**.**isnull**().sum())**

As we can see the due to popularity of the dataset sourced from UCI Machine Learning Repository there are no missing or null values in the dataset as it was arguably cleaned beforehand. As the dataset is filled with 23 categorical variables, we want to convert them

into some form of numerical value so that they can easily fit into some form of model. The target variable for this dataset is defined as ‘class’ which contains the information on whether a mushroom is poisonous or not, and will be represented in a binary form, ‘1’ for poisonous and ‘0’ for edible.

Problem to be addressed

If we were to look at the problem in detail, we can identify that the aim of the project is not only to build and evaluate a machine learning model but to effectively/comparatively find the most efficient model that is best suited to solve the real-world problem in our dataset. The dataset itself is structured in a way where classification models are the best and only way to deal with a discrete numeric problem. As we are evaluating multiple machine learning models to better solve the problem, a fitness function is needed so that we can gain some form of measurement that’ll allow us to check which machine learning model was better equipped to solve the problem. For this dataset we will be using the classification accuracy metric to compare our machine learning models.

Machine Learning Model

Before we can enter the dataset into a principal component analysis form it is best to first analyze the correlation coefficients. A correlation coefficient is simply a measure of direction and strength of the linear relationship among variables.

Chart, treemap chart

Description automatically generated

Since our dataset has a lot of dimensions and input features, we want to find a way that allows us to interpret and visualise the data better. To do this, we will use a method called principal component analysis(PCA). In short PCA will allow us to reduce the dimensionality of the data, whilst minimizing any loss of information. However, PCA does have the potential to cause a loss of some spatial information which is always important for classification. We are also approaching the problem with a multitude of classification algorithms these will be:

Logistic Regression, Decision Tree Classifier, Random Forest Classifier, K-Nearest Neighbours(K-NN), Support Vector, and Naïve Bayes Classifier

Once the categorical variables have been encoded to numerical variables, we’ll apply PCA to the dataset and we’ll split the dataset into an x train, x test, y train and y test using the ‘Sklearn’ package. It should also be noted that before the data is fitted into test and train data, we want to standardize the features so that we can easily compare the scores or evaluation of the models; it will also possibly remove any bias that may be lingering from the numeric values.

*\*See Standardize Function Below\**

After applying the standardization on the predictor features, we will then be creating functions that allows us to plot and visualise the data. We shall also create an evaluation function that will use the classification accuracy metric and Standard Deviation to compare the models. It should also be noted that when training the models, we used a test size of 33% to give the machine an optimal amount of information that won’t hinder or create bias for the outcome.

To visualise the classification, we want to make sure, we can interpret the model so we will use PCA to create a graph in two dimensions that separate the mushrooms into edible and poisonous.

In this project we did not use any type of feature engineering, or feature selection as we tried to incorporate as many as the base features as possible. However, during the report we could of created some new non-linear features for the model to use aswell, but this would create such problems as over-fitting and structual multi-colinearity.

Chart, scatter chart

Description automatically generated

Figure 1 – PCA

Once we had created our visualisation function and our evaluation function, we could finally create our models and put them into the function. The first model that was tested was the Logistic Regression classification model.

Chart

Description automatically generated

Figure 2 - Logistic Regression

Logistic Regression is a method of classification that typically looks at linearly separating the target feature, due to the fact there were only 2 different labels there was only one line. The visualisation above shows how the trained model could perform on the test data for the mushroom dataset. When the model was completed the evaluation function then showed the accuracy score.

#-----------------------------------#

Test Results for Logistic Regression

Accuracy Score:

0.9103889709502708

Cross Validation Score:

[0.91222313 0.90894176 0.89991797 0.9047619 0.88998358]

Mean Score:

0.9031656678399343

Standard Deviation:

0.007775138185715516

#-----------------------------------#Chart

Description automatically generated

Chart

Description automatically generated with medium confidence

Figure 3 - Support Vector Classification

SVC is the next method of classification that was completed by the machine. SVC is known to be able to solve both linear and nonlinear problems. By looking at figure 3 we can immediately notice that the model that was outputted was not a straight line and was able to yield a slightly better accuracy score.

#-----------------------------------#

Test Results for SVC

Accuracy Score:

0.9172821270310192

Cross Validation Score:

[0.91304348 0.91386382 0.90894176 0.91050903 0.8998358 ]

Mean Score:

0.9092387768379961

Standard Deviation:

0.005019653618834548

#-----------------------------------#Chart

Description automatically generated

Chart

Description automatically generated

Figure 4 - Random Forest

Random Forest is a method of classification which is highly used for its diversity and can also be used for regression-based tasks. Random forest essentially has the general idea of creating an array of decision trees and merge them together. When using the Random Forest model in the mushroom dataset it proved to be one of the more effective one’s at predicting the target class

#-----------------------------------#

Test Results for Random Forest

Accuracy Score:

0.9320531757754801

Cross Validation Score:

[0.9392945 0.91468417 0.93273175 0.92446634 0.92118227]

Mean Score:

0.9264718045290022

Standard Deviation:

0.008655148764689159

#-----------------------------------#

Chart, treemap chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Figure 5 - Decision Tree

Decision tree classification is one of the simplest ways of classification and it essentially breaks down a dataset into smaller subsets. A decision tree will consist of nodes, branches, and leaf nodes. Decision trees are also known for their extremely fast classifying speeds and ability to still be easy to interpret. However, a Decision Tree model also has the potential to be over-fit. The accuracy score of this model was

#-----------------------------------#

Test Results for DecisionTree Classifier

Accuracy Score:

0.9005416051206302

Cross Validation Score:

[0.91304348 0.89663659 0.8949959 0.89244663 0.89326765]

Mean Score:

0.8980780499238252

Standard Deviation:

0.007620948693101168

#-----------------------------------#Chart

Description automatically generated

Chart

Description automatically generated

Figure 6 - Naive Bayes

The Naïve Bayes classifier method is a classification algorithm that works on a principle of conditional probability. The classification method was able to quickly classify the targets, however, is more suited to solving categorical input variables than the numerical ones the model used. The ending result was just over 90% accuracy. When attempting to run this machine learning model it was notably longer than any of the others; most likely due to the numerous calculations it must make; compared to that of logistic regression for example.

#-----------------------------------#

Test Results for Naive Bayes

Accuracy Score:

0.9030034465780403

Cross Validation Score:

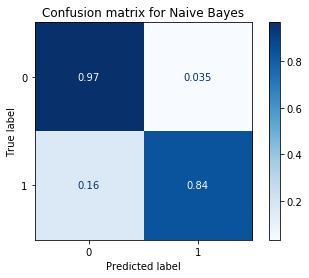
[0.89991797 0.90319934 0.89417555 0.8908046 0.88423645]

Mean Score:

0.8944667827811162

Standard Deviation:

0.00669520830706812

#-----------------------------------#

Map

Description automatically generated

Figure 7 - K Nearest Neighbors (K-NN)

The final classification method that was used was the K Nearest Neighbors algorithm, which is known for its popularity and effectiveness. From training and running the model it was able to produce the highest accuracy score out of all 7 algorithms. This model was one of the fastest to produce due to its instance based learning method which means that it does not require a training method, however K-NN is susceptible/sensitive to noisy or missing data; thus the dataset being used should be properly cleaned before-hand  
#-----------------------------------#

Test Results for K Nearest Neighbors (K-NN)

Accuracy Score:

0.9330379123584441

Cross Validation Score:

[0.93683347 0.92206727 0.92698934 0.92528736 0.92528736]

Mean Score:

0.9272929572949374

Standard Deviation:

0.005029239832140231

#-----------------------------------#

Chart, treemap chart

Description automatically generated

|  |  |
| --- | --- |
| Classification Algorithm | Accuracy Score |
| Logistic Regression | 0.9103 |
| SVC | 0.9172 |
| Random Forest | 0.9305 |
| Decision Tree | 0.9005 |
| Naïve Bayes | 0.9030 |
| K-NN | 0.9320 |

When looking at the results of the machine learning models, the models were able to stay within a 5% bracket of predictive accuracy between each other. It should also be noted that the average mean percentage of the algorithms was 0.9155. A significant evaluation that could also be made is that Random Forest and Decision Tree classifiers have a 3% different even when they are similar in terms of design and processing speed. The results shown are what the evaluation code outputted; however, these results were not tested for overfitting which could have impacted the results to show a higher accuracy. Ultimately the results were able to prove that the machine learning models make an effective way to predict real world problems within a decent percentile accuracy. These results although were only tested through one run of the program could be changed because we didn’t use a ‘random seed’ to create re-producible results. However, through repeated testing they do show similar percentages.

Conclusion

In conclusion it can be said that throughout the project we effectively tested multiple classification algorithms and made a comparative evaluation for which one was the most effective; in the case of this dataset, it was K-Nearest Neighbors. Ultimately the report conveyed how not one classification model may be suitable for all classification models, the report should enforce the idea of testing these multiple algorithms to find the most optimal one.

With the report complete, one would be able to assume that with the results shown, the dataset classification problem was accurately solved with a maximum of 93% prediction and further proves the notion of machine learning models being able to solve real world problems with ease.

If the report was to be completed again to gain further insight into building classifications models, one change that would be made is to include more than one dataset. Another change that would be significant to include is measuring the effects of over-fitting and feature engineering in classification models. In summary, it can be acknowledged that the report was successful in its evaluation of machine learning models.

# References

Pedregosa, F. et al., 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct), pp.2825–2830

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI:

McKinney, W. & others, 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*. pp. 51–56.

Hunter, J.D., 2007. Matplotlib: A 2D graphics environment. *Computing in science &amp; engineering*, 9(3), pp.90–95.

Code

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

**from** sklearn**.**decomposition **import** PCA

**from** sklearn**.**metrics **import** plot\_confusion\_matrix

**from** sklearn**.**model\_selection **import** cross\_val\_score

**from** sklearn**.**model\_selection **import** train\_test\_split

**from** sklearn**.**preprocessing **import** LabelEncoder

**from** sklearn**.**preprocessing **import** StandardScaler

**from** sklearn**.**neighbors **import** KNeighborsClassifier **as** KNN

**from** sklearn**.**ensemble **import** RandomForestClassifier

**from** sklearn**.**metrics **import** accuracy\_score

**from** sklearn**.**linear\_model **import** LogisticRegression

**from** sklearn**.**tree **import** DecisionTreeClassifier **as** DT

**from** sklearn**.**svm **import** SVC

**from** sklearn**.**naive\_bayes **import** GaussianNB **as** NB

**from** matplotlib**.**axes**.**\_axes **import** \_log **as** matplotlib\_axes\_logger

**from** matplotlib**.**colors **import** ListedColormap

**import** seaborn **as** sns

matplotlib\_axes\_logger**.**setLevel**(**'ERROR'**)**

df **=** pd**.**read\_csv**(**"mushrooms.csv"**)**

**print(**df**.**shape**)**

**print(**df**.**head**)**

**print(**df**.**isnull**().sum())**

df **=** df**.**drop**(**'veil-type'**,**axis**=**1**)**

#Seperating target variable and predictor variables.

X**=**df**.**drop**(**'class'**,**axis**=**1**)**

y**=**df**[**'class'**]**

np**.**random**.**seed**(**40**)**

#Enconde the data, change catagorical to numbers

Encoder\_X **=** LabelEncoder**()**

**for** col **in** X**.**columns**:**

X**[**col**]** **=** Encoder\_X**.**fit\_transform**(**X**[**col**])**

Encoder\_y**=**LabelEncoder**()**

y **=** Encoder\_y**.**fit\_transform**(**y**)**

corr\_matrix **=** X**.**corr**()**

**print(**corr\_matrix**)**

plt**.**figure**(**figsize**=(**25**,**15**))**

plt**.**title**(**'Correlation Heatmap of Mushroom Dataset'**)**

a **=** sns**.**heatmap**(**corr\_matrix**,** square**=True,** annot**=True,** fmt**=**'.2f'**,** linecolor**=**'black'**)**

a**.**set\_xticklabels**(**a**.**get\_xticklabels**(),** rotation**=**90**)**

a**.**set\_yticklabels**(**a**.**get\_yticklabels**(),** rotation**=**30**)**

plt**.**show**()**

#Get Dummy Data for X features

X**=**pd**.**get\_dummies**(**X**,**columns**=**X**.**columns**,**drop\_first**=True)**

#Standardize the data

sc **=** StandardScaler**()**

X**=**sc**.**fit\_transform**(**X**)**

#Apply PCA

**print(type(**X**))**

pca **=** PCA**(**n\_components**=**2**)**

components **=** pca**.**fit\_transform**(**X**)**

#Train the data

X\_train**,** X\_test**,** y\_train**,** y\_test **=** train\_test\_split**(**components**,** y**,** random\_state**=**0**)**

#Put PCA into a pandas dataframe

pca\_df **=** pd**.**DataFrame**(**data **=** components

**,** columns **=** **[**'principal component 1'**,** 'principal component 2'**])**

#Check Tail of PCA

**print(**pca\_df**.**tail**())**

#plot PCA

plt**.**figure**()**

plt**.**figure**(**figsize**=(**10**,**15**))**

plt**.**yticks**(**fontsize**=**25**)**

plt**.**xticks**(**fontsize**=**25**)**

plt**.**xlabel**(**'PC 1'**,**fontsize**=**20**)**

plt**.**ylabel**(**'PC 2'**,**fontsize**=**20**)**

plt**.**title**(**"Principal Component Analysis of Mushrooms"**,**fontsize**=**20**)**

targets **=** **[**'p'**,** 'e'**]**

colors **=** **[**'purple'**,** 'g'**]**

**for** target**,** color **in** **zip(**targets**,**colors**):**

indicesToKeep **=** df**[**'class'**]** **==** target

plt**.**scatter**(**pca\_df**.**loc**[**indicesToKeep**,** 'principal component 1'**]**

**,** pca\_df**.**loc**[**indicesToKeep**,** 'principal component 2'**],** c **=** color**,** s **=** 60**)**

plt**.**legend**(**targets**,**prop**={**'size'**:** 20**})**

#create visulisation function

########

**def** mushroom\_test**(**model**,**model\_name**):**

plt**.**figure**(**figsize**=(**10**,**15**))**

X\_set**,** y\_set **=** X\_test**,** y\_test

X1**,** X2 **=** np**.**meshgrid**(**np**.**arange**(**start **=** X\_set**[:,** 0**].min()** **-** 1**,** stop **=** X\_set**[:,** 0**].max()** **+** 1**,** step **=** 0.01**),**

np**.**arange**(**start **=** X\_set**[:,** 1**].min()** **-** 1**,** stop **=** X\_set**[:,** 1**].max()** **+** 1**,** step **=** 0.01**))**

plt**.**contourf**(**X1**,** X2**,** model**.**predict**(**np**.**array**([**X1**.**ravel**(),** X2**.**ravel**()]).**T**).**reshape**(**X1**.**shape**),**

alpha **=** 0.6**,** cmap **=** ListedColormap**((**'purple'**,** 'green'**)))**

plt**.**xlim**(**X1**.min(),** X1**.max())**

plt**.**ylim**(**X2**.min(),** X2**.max())**

**for** i**,** j **in** **enumerate(**np**.**unique**(**y\_set**)):**

plt**.**scatter**(**X\_set**[**y\_set **==** j**,** 0**],** X\_set**[**y\_set **==** j**,** 1**],**

c **=** ListedColormap**((**'purple'**,** 'green'**))(**i**),** label **=** j**)**

plt**.**title**(**"%s Test Set" **%(**model\_name**))**

plt**.**xlabel**(**'PC 1'**)**

plt**.**ylabel**(**'PC 2'**)**

plt**.**legend**()**

#Create evaulation function

**def** evaulation**(**model**,**X\_test**,**y\_test**,**model\_name**):**

**print(**"#-----------------------------------#"**)**

**print(**"Test Results for " **+** model\_name**)**

**print(**"Accuracy Score: "**)**

**print(**accuracy\_score**(**y\_test**,**model**.**predict**(**X\_test**)))**

**print(**"Cross Validation Score: "**)**

cv\_acc**=**cross\_val\_score**(**model**,** X\_train**,**y\_train**,**cv**=**5**)**

**print(**cv\_acc**)**

**print(**"Mean Score: "**)**

**print(**cv\_acc**.**mean**())**

**print(**"Standard Deviation: "**)**

**print(**cv\_acc**.**std**())**

**print(**"#-----------------------------------#"**)**

#Logistic Regression test model

log\_model **=** LogisticRegression**()**

fitted\_log **=** log\_model**.**fit**(**X\_train**,**y\_train**)**

fitted\_log

mushroom\_test**(**log\_model**,**'Logistic Regression'**)**

evaulation**(**log\_model**,** X\_test**,** y\_test**,** 'Logistic Regression'**)**

#confusion matrix

matrix **=** plot\_confusion\_matrix**(**fitted\_log**,** X\_test**,** y\_test**,**

cmap**=**plt**.**cm**.**Blues**,**

normalize**=**'true'**)**

plt**.**title**(**'Confusion matrix for Logistic Regression'**)**

plt**.**show**(**matrix**)**

plt**.**show**()**

#SVC test model

svc\_model **=** SVC**(**kernel**=**'rbf'**,**random\_state**=**42**)**

fitted\_svm **=** svc\_model**.**fit**(**X\_train**,**y\_train**)**

fitted\_svm

mushroom\_test**(**svc\_model**,**'SVC'**)**

evaulation**(**svc\_model**,** X\_test**,** y\_test**,** 'SVC'**)**

#confusion matrix

matrix **=** plot\_confusion\_matrix**(**fitted\_svm**,** X\_test**,** y\_test**,**

cmap**=**plt**.**cm**.**Blues**,**

normalize**=**'true'**)**

plt**.**title**(**'Confusion matrix for SVM'**)**

plt**.**show**(**matrix**)**

plt**.**show**()**

#DT test model

dt\_model **=** DT**(**criterion**=**'entropy'**,**random\_state**=**42**)**

fitted\_dt **=** dt\_model**.**fit**(**X\_train**,**y\_train**)**

fitted\_dt

mushroom\_test**(**dt\_model**,** 'DecisionTree Classifier'**)**

evaulation**(**dt\_model**,** X\_test**,** y\_test**,** 'DecisionTree Classifier'**)**

#confusion matrix

matrix **=** plot\_confusion\_matrix**(**fitted\_dt**,** X\_test**,** y\_test**,**

cmap**=**plt**.**cm**.**Blues**,**

normalize**=**'true'**)**

plt**.**title**(**'Confusion matrix for Decision Tree'**)**

plt**.**show**(**matrix**)**

plt**.**show**()**

#KNN test model

knn\_model **=** KNN**()**

fitted\_knn **=** knn\_model**.**fit**(**X\_train**,**y\_train**)**

fitted\_knn

mushroom\_test**(**knn\_model**,** 'K Nearest Neighbors (K-NN)'**)**

evaulation**(**knn\_model**,** X\_test**,** y\_test**,** 'K Nearest Neighbors (K-NN)'**)**

#confusion matrix

matrix **=** plot\_confusion\_matrix**(**fitted\_knn**,** X\_test**,** y\_test**,**

cmap**=**plt**.**cm**.**Blues**,**

normalize**=**'true'**)**

plt**.**title**(**'Confusion matrix for KNN'**)**

plt**.**show**(**matrix**)**

plt**.**show**()**

#NB test model

nb\_model **=** NB**()**

fitted\_nb **=** nb\_model**.**fit**(**X\_train**,**y\_train**)**

fitted\_nb

mushroom\_test**(**nb\_model**,** 'Naive Bayes'**)**

evaulation**(**nb\_model**,** X\_test**,** y\_test**,** 'Naive Bayes'**)**

#confusion matrix

matrix **=** plot\_confusion\_matrix**(**fitted\_nb**,** X\_test**,** y\_test**,**

cmap**=**plt**.**cm**.**Blues**,**

normalize**=**'true'**)**

plt**.**title**(**'Confusion matrix for Naive Bayes'**)**

plt**.**show**(**matrix**)**

plt**.**show**()**

#RF test model

rf\_model **=** RandomForestClassifier**(**n\_estimators **=** 50**,** criterion **=** 'entropy'**,** random\_state **=** 42**)**

fitted\_rf **=** rf\_model**.**fit**(**X\_train**,** y\_train**)**

fitted\_rf

mushroom\_test**(**rf\_model**,** 'Random Forest'**)**

evaulation**(**rf\_model**,** X\_test**,** y\_test**,** 'Random Forest'**)**

#confusion matrix

matrix **=** plot\_confusion\_matrix**(**fitted\_rf**,** X\_test**,** y\_test**,**

cmap**=**plt**.**cm**.**Blues**,**

normalize**=**'true'**)**

plt**.**title**(**'Confusion matrix for Random Forest'**)**

plt**.**show**(**matrix**)**

plt**.**show**()**