Comparing Support Vector Classification and Decision Tree classifier in the prediction of Pizza brands

Spring Semester, 2022

1. Introduction

As it is known to us, pizza is the food, which consists of many ingredients. In fact, pizza also attracts the appetite from people, and there are many pizza restaurants all over the world.

The application domain could be used in the commercial field for business owners to analyze their pizzas, in order to have a thorough understanding of the markets. Since it is a course project in the Machine Learning field, it is expected to explore how the machine-learning methods could be applied to the realistic problems. Here we want to predict the pizza brands based on the given dataset. Moreover, this topic enables people to know the brand of pizza they want to buy, if the information of pizza has been provided. Then people would know better about the pizza brands, which helps them to decide the brand they want to purchase.

Regarding the structure of the report, the clarification on this machine learning problem is presented in the Section 2 “Problem Formulation”, while the methods will be presented in the Section 3. Then the Section 4 and Section 5 are expected to show the result and draw the conclusion of the project respectively.

2. Problem Formulation

In terms of the problem of project, we could start to assume that one consumer knows the size, the taste and the price of the pizza, but this person is unaware of the brand of this pizza. Then problem arises: What is the brand of the pizza?

This dataset comes from the online open-source platform Kaggle [1]. The data points of the problem are the record of one pizza, which includes the information of its original company name, the pizza name, the type, the size, as well as the price. The total number of the data numbers is 371. Since the information except the price from the dataset are not in the form of numbers, we could use numerical values to indicate the information. For example, since there are only four brands of the pizza (Domino, Pizza Hut, Godfather, IMO), the numbers from 1 to 4 are used to indicate the brand of pizza respectively. Therefore, it could be a multiclass classification problem, where there are 4 potential categories of outcomes.

2.1 Labels and Features of the data points

This part will introduce the labels and features of the adopted data points in this project briefly.

Label: The brand of the pizza.

Candidate Features: The type, the size, and the price of the pizza. (3 features)

3. Methods

3.1 Dataset

To get the data points of this project, I collect the data from the Kaggle platform [1], and each data point represent the indicators in one record for each kind of the pizza, which is also mentioned in the "Problem Formulation" section. There are 371 data points from 4 pizza companies in total, with 0 missing data in the dataset.

As for the candidate feature, I will consider three features including the type, the size, and the price of the pizza. Since the original value of these three values is in the format of text, I will convert the text into discrete integers. For example, there are several sizes of the pizza, and I will use the number 1 to indicate the size of "Mini", the number 2 to indicate the size of "Small", and the largest number 6 to the size of "Jumbo".

Similarly, as I select the brand of the pizza as my label, since there are 4 brands in total, I will use **the number 0 to 3** to indicate these 4 brands of pizza restaurants.

3.2 Feature Selection

Firstly, I will present the visualization of my data with scatterplots.

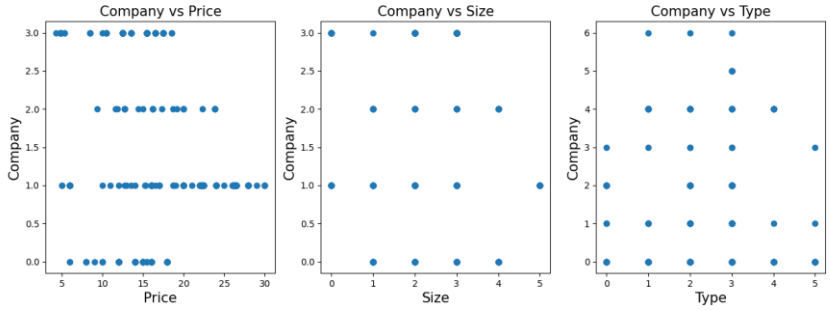

Figure 1. Data visualization

From Figure 1, we could notice that the distribution of the price of the pizza is most sparse, which means that it would be easier for the model to judge the brand of the pizza. For example, for the pizza price over 25 dollars, we could find that it belongs to the brand with the label '1'. Thus, among the candidate features, the "Price" would be adopted as the feature of the dataset.

3.3 The motivation of selected methods and models

3.3.1 The first model of the project

Since this is a multi-class problem, the classification models from the method Support Vector Machine (SVM) could be used. It is known that the Logistic Regression methods focus more on the classification from 2 classes, and the Scikit Learn library [2] has provided classes including the SVC (Support Vector Classification) for the classification task, with the use of decision function. The decision function could help to the muti-class classification case [2].

Now that the SVC is a class of SVM classification methods, it uses basic linear maps $h(\boldsymbol{x}) = w^T \boldsymbol{x}$ [3], which helps to finish the task of classification.

3.3.2 The second method of the project

Apart from the SVC, another suitable model for the classification comes from the method Decision Tree. The Scikit Learn library also provided the decision tree classifier for the multi-class classification task [4]. It could be found that the decision tree enables to deal with the classification into multiple classes [5], which is suitable for the task of the project.

Since the model is based on the decision tree method, it uses tree structure [5] to finish the task of classification into different categories.

3.4 Loss function

3.4.1 The first model of the project

Since the model from the SVM method has been selected, the **Hinge Loss** would be used, which is "L((***x***, y), h) = max{0, 1-yh(***x***)}" [6]. And the Hinge Loss is chosen because this loss function is the inevitable basis in the SVM method [3], which means that the SVM uses the Hinge Loss with the regularized parameters [3]. And the Hinge loss would perform the task of learning the robust hypothesis [3].

3.4.2 The second model of the project

Because I select the model from the decision tree method, according to the specification [4], either Gini impurity or information gain could be adopted as the loss function. Considering the feasibility of the function, I finally select the **information gain** as the loss function. According to the knowledge of the information theory, the entropy of the variable X is defined as $-\sum p \log(p)$[7], which could be used as the information gain of the tree decision model.

3.5 Size of training and validation dataset

As mentioned in the Section 3.1, there are 371 data points of my dataset from the Kaggle [1]. In general, the data records are quite enough for me to process. Thus, I select 320 of data points from the entire dataset, and use the single split into training and validation set.

For both the first model based on SVC and the second model based on the decision tree, I select 80% of the entire data points as my training dataset, while the rest (20%) of the data points will become the validation dataset. Since allocating more data to the training dataset helps to fit the model, it is reasonable to select 80% of the 320 data points as the training dataset.

4. Results (with visualization)

4.1 Training Error and Validation Error

4.1.1 The first model based on SVC method

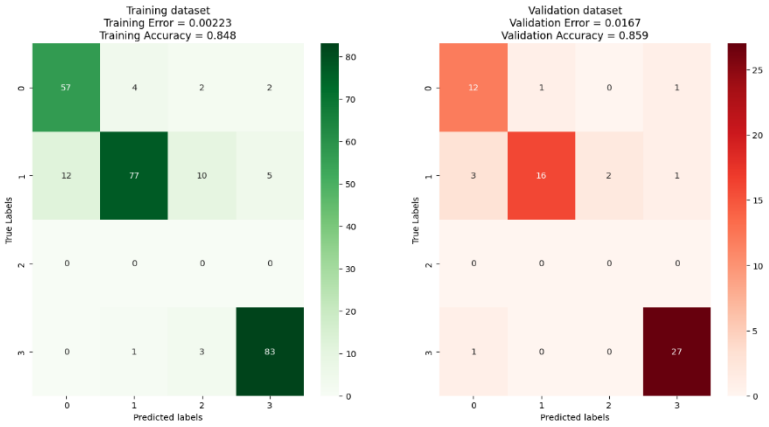

Figure 2. The training and validation error of the model based on SVC

We could find that the training error and validation error are around 0.002 and 0.017 respectively. Additionally, the training and validation accuracy is 0.847 and 0.859 respectively.

4.1.2 The second model based on the decision tree method

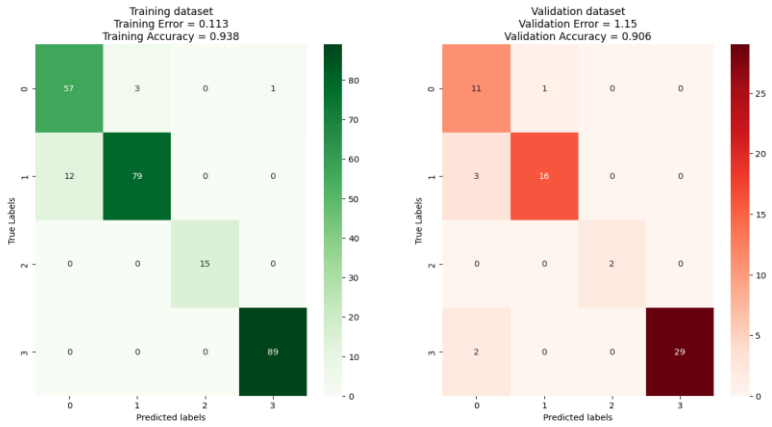

Figure 3. The training and validation error of the model based on decision tree

We could see that the training error and validation error are around 0.113 and 1.15 respectively. Additionally, the training and validation accuracy is 0.938 and 0.906 respectively.

Therefore, according to indicators including the training and validation error, as well as training and testing accuracy, I would finally select the model **based on the SVC method**.

4.2 Testing set and Testing error

In terms of the testing set, the data points for the training and validation set come from the 320 data points, which are randomly selected from the entire 371 datapoints. Thus, there are some data points left which are not used before, and I select 32 data points for testing set.

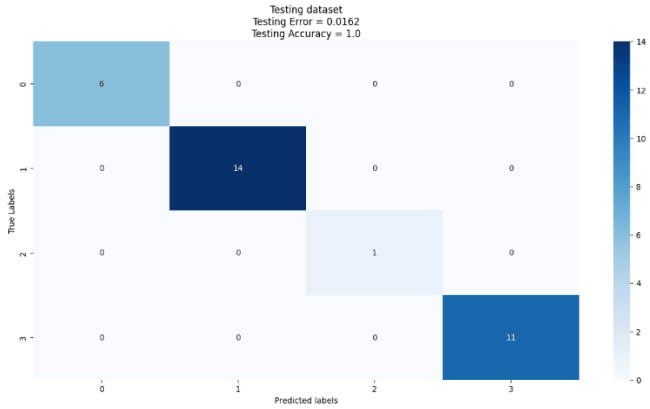

Figure 4. The testing error from the model based on SVC method

From Figure 4, the testing error for the model is 0.0162, with the testing accuracy of 100%.

5. Conclusion

In general, the model built on SVC method performs better than the decision tree model, due to the lower training and validation error. The testing error and accuracy are also presented. The classification result is quite satisfying and meet the previous expectation.

As for the potential improvements, though there are 371 the datapoints in this project, if the size of the dataset is larger. Also, as for the decision tree model, the information gain is selected in the loss function, so another loss function Gini impurity could be considered for further work. Additionally, to have a more convincing result, the candidate features of the dataset could be more, which could be better for the choice of deciding which feature to use.

6. References

[1] https://www.kaggle.com/knightbearr/analysis-pizza-price-data-knightbearr/data
[2] Support Vector Classification, Scikit Learn, https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[3] A. Jung, “Machine Learning: basics”, 2022, pp.90-92, https://github.com/alexjungaalto/MachineLearningTheBasics/blob/master/MLBasicsBook.pdf
[4] Decision Tree Classifier, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier
[5] Decision Trees, https://scikit-learn.org/stable/modules/tree.html
[6] A. Jung, “Machine Learning: basics”, 2022, pp.62-63, https://github.com/alexjungaalto/MachineLearningTheBasics/blob/master/MLBasicsBook.pdf
[7] C. E. Shannon, "A mathematical theory of communication", The Bell System Technical Journal, vol. 27, no. 3, pp. 379-423, 1948

Appendix

1. The codes for this project are presented in the next page for reference:

ML_Project

March 31, 2022

[ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import hinge_loss,log_loss
from sklearn import metrics

def SVC_1(name):
    # Load the dataset
    PizzaData = pd.read_csv(name)
    print("The original size of the dataset:")
    print(PizzaData.shape)
    print("The column of the dataset:")
    print(PizzaData.columns)
    X0= PizzaData[['Company', 'Type']]
    print(X0.shape)

    # Visualize dataset
    fig, axes = plt.subplots(1, 3, figsize=(21, 5))  # Create the plot with 3␣
 ↪subplots to show the relationship between the label and the feature
    axes[0].scatter(PizzaData['Price'],
                    PizzaData['Company'])
    axes[0].set_xlabel("Price", size=15)
    axes[0].set_ylabel("Company", size=15)
    axes[0].set_title("Company vs Price", size=16)

    axes[1].scatter(PizzaData['Size'],
                    PizzaData['Company'])
    axes[1].set_xlabel("Size", size=15)
    axes[1].set_ylabel("Company", size=15)
    axes[1].set_title('Company vs Size', size=16)
```

```
    axes[2].scatter(PizzaData['Size'],
                    PizzaData['Type'])
    axes[2].set_xlabel("Type", size=15)
    axes[2].set_ylabel("Company", size=15)
    axes[2].set_title('Company vs Type', size=16)
    plt.show()

    X = PizzaData['Price'].to_numpy().reshape(-1, 1)
    y = PizzaData['Company'].to_numpy()

    print(X.shape)
    print(y.shape)

    X_clf = np.copy(X)
    y_clf = y
    np.random.seed(500)
    idx = np.random.choice(np.arange(371), 320)  # choose 320 datapoints from␣
↪the entire dataset
    print(len(idx))

    #Split the dataset
    X_train, X_val, y_train, y_val = train_test_split(X_clf[idx, :],␣
↪y_clf[idx], test_size=0.2,
                                                      ␣
↪random_state=42)
    c = 100000

    #Select the SVC Model
    clf_1 = SVC(C=c, decision_function_shape="ovr")

    #Start to fit the model
    clf_1.fit(X_train, y_train)
    y_pred_train = clf_1.predict(X_train)
    X_train0 = np.array(X_train).reshape(256, 1)
    X_val0 = np.array(X_val).reshape(64, 1)
    y_pred_train0 = np.array(y_pred_train)

    pred_decision = clf_1.decision_function(X_train0)
    tr_accs = accuracy_score(y_train, y_pred_train)  # Calculate the training␣
↪accuracy
    labels = np.array([0, 1, 2, 3])
    tr_error1 = hinge_loss(y_pred_train0, pred_decision, labels=labels)  #␣
↪Calculate the training error
    pred_decision1 = clf_1.decision_function(X_val0)

    y_pred_val = clf_1.predict(X_val)
    y_pred_val0 = np.array(y_pred_val)
```

```
    val_error = hinge_loss(y_pred_val0, pred_decision1, labels=labels)
    val_accs = accuracy_score(y_val, y_pred_val)  # Calculate the validation␣
↪accuracy

    # Visualize the result
    confusion_matrix_result = metrics.confusion_matrix(y_pred_train, y_train)
    confusion_matrix_result1 = metrics.confusion_matrix(y_pred_val, y_val)

    plt.figure(1, figsize=(16, 12))
    plt.subplot(1, 2, 1)
    sns.heatmap(confusion_matrix_result, annot=True, cmap='Greens')
    plt.xlabel('Predicted labels')
    plt.ylabel('True Labels')
    plt.title(f'Training dataset\nTraining Error = {tr_error1:.3}\nTraining␣
↪Accuracy = {tr_accs:.3}')

    plt.subplot(1, 2, 2)
    sns.heatmap(confusion_matrix_result1, annot=True, cmap='Reds')
    plt.xlabel('Predicted labels')
    plt.ylabel('True Labels')
    plt.title(f'Validation dataset\nValidation Error = {val_error:.
↪3}\nValidation Accuracy = {val_accs:.3}')
    plt.show()


    X_test = []
    y_test = []
    id_rest = []  # Store the index of the data points which are not used in␣
↪the training and validation process

    for i in range(371):
        if i not in idx:
            id_rest.append(i)
    # Define the size of testing dataset
    test_size = 32
    idr = np.random.choice(id_rest, test_size)

    X_test, y_test = X_clf[idr, :], y_clf[idr]

    #Start to fit the testing dataset
    clf_1.fit(X_test, y_test)
    y_pred_test = clf_1.predict(X_test)
    X_test0 = np.array(X_test).reshape(test_size, 1)
    y_pred_test0 = np.array(y_pred_test)

    pred_decision2 = clf_1.decision_function(X_test0)
```

```
    test_accs = accuracy_score(y_test, y_pred_test)  # Calculate the testing␣
↪accuracy
    labels = np.array([0, 1, 2, 3])
    test_error1 = hinge_loss(y_pred_test0, pred_decision2, labels=labels)  #␣
↪Calculate the testing error

    confusion_matrix_result = metrics.confusion_matrix(y_pred_test, y_test)
    plt.figure(figsize=(16, 12))
    sns.heatmap(confusion_matrix_result,annot=True,cmap='Blues')
    plt.xlabel('Predicted labels')
    plt.ylabel('True Labels')
    plt.title(f'Testing dataset\nTesting Error = {test_error1:.3}\nTesting␣
↪Accuracy = {test_accs:.3}')
    plt.show()

def decision_tree(name):
    # Load the dataset
    PizzaData = pd.read_csv(name)
    print("The original size of the dataset:")
    print(PizzaData.shape)
    print("The column of the dataset:")
    print(PizzaData.columns)
    X0= PizzaData[['Company', 'Type']]
    print(X0.shape)

    # Visualize dataset
    fig, axes = plt.subplots(1, 3, figsize=(21, 5))  # Create the plot with 3␣
↪subplots to show the relationship between the label and the feature
    axes[0].scatter(PizzaData['Price'],
                    PizzaData['Company'])
    axes[0].set_xlabel("Price", size=15)
    axes[0].set_ylabel("Company", size=15)
    axes[0].set_title("Company vs Price", size=16)

    axes[1].scatter(PizzaData['Size'],
                    PizzaData['Company'])
    axes[1].set_xlabel("Size", size=15)
    axes[1].set_ylabel("Company", size=15)
    axes[1].set_title('Company vs Size', size=16)

    axes[2].scatter(PizzaData['Size'],
                    PizzaData['Type'])
    axes[2].set_xlabel("Type", size=15)
    axes[2].set_ylabel("Company", size=15)
    axes[2].set_title('Company vs Type', size=16)
    plt.show()
```

```
   X = PizzaData['Price'].to_numpy().reshape(-1, 1)
   y = PizzaData['Company'].to_numpy()

   print(X.shape)
   print(y.shape)

   X_clf = np.copy(X)
   y_clf = y
   np.random.seed(500)
   idx = np.random.choice(np.arange(371), 320)  # choose 320 datapoints from␣
↪the entire dataset
   print(len(idx))

   # Set up the training dataset and validation dataset
   X_train, X_val, y_train, y_val = train_test_split(X_clf[idx, :],␣
↪y_clf[idx], test_size=0.2, random_state=42)
   clf_2 = DecisionTreeClassifier(criterion='entropy')
   #Train the model
   clf_2.fit(X_train, y_train)
   y_pred_train = clf_2.predict(X_train)
   tr_accs = accuracy_score(y_train, y_pred_train)  # calculate the training␣
↪accuracy

   y_pred_val = clf_2.predict(X_val)
   val_accs = accuracy_score(y_val, y_pred_val)  # calculate the validation␣
↪accuracy

   y_pred_train_prob = clf_2.predict_proba(X_train)
   y_pred_val_prob = clf_2.predict_proba(X_val)

   tr_errs = log_loss(y_train, y_pred_train_prob)  # Calculate the training␣
↪error

   val_errs = log_loss(y_val, y_pred_val_prob)  # Calculate the validation␣
↪error

   # Visualize the result
   confusion_matrix_result = metrics.confusion_matrix(y_pred_train, y_train)
   confusion_matrix_result1 = metrics.confusion_matrix(y_pred_val, y_val)

   plt.figure(1, figsize=(16, 12))
   plt.subplot(1, 2, 1)
   sns.heatmap(confusion_matrix_result, annot=True, cmap='Greens')
   plt.xlabel('Predicted labels')
   plt.ylabel('True Labels')
```

```
    plt.title(f'Training dataset\nTraining Error = {tr_errs:.3}\nTraining␣
↪Accuracy = {tr_accs:.3}')

    plt.subplot(1, 2, 2)
    sns.heatmap(confusion_matrix_result1, annot=True, cmap='Reds')
    plt.xlabel('Predicted labels')
    plt.ylabel('True Labels')
    plt.title(f'Validation dataset\nValidation Error = {val_errs:.
↪3}\nValidation Accuracy = {val_accs:.3}')
    plt.show()
    id_rest = []

    # Check the index of data points which has not been used in the training␣
↪and validation dataset
    for i in range(371):
        if i not in idx:
            id_rest.append(i)

    # Define the size of testing dataset
    test_size = 32
    idr = np.random.choice(id_rest, test_size)
    # Start to test the remaining dataset
    X_test, y_test = X_clf[idr, :], y_clf[idr]

    clf_2.fit(X_test, y_test)
    y_pred_test = clf_2.predict(X_test)
    y_pred_test_prob = clf_2.predict_proba(X_test)

    test_accs = accuracy_score(y_test, y_pred_test)  # calculate the testing␣
↪accuracy
    test_errs = log_loss(y_test, y_pred_test_prob) # calculate the testing error

    #Visualize the result
    confusion_matrix_result2 = metrics.confusion_matrix(y_pred_test, y_test)
    plt.figure(figsize=(16, 12))
    sns.heatmap(confusion_matrix_result2, annot=True, cmap='Blues')
    plt.xlabel('Predicted labels')
    plt.ylabel('True Labels')
    plt.title(f'Testing dataset\nTesting Error = {test_errs:.3}\nTesting␣
↪Accuracy = {test_accs:.3}')
    plt.show()

def main():
    SVC_1('processed_data.csv')
    decision_tree('processed_data.csv')

# Press the green button in the gutter to run the script.
```

```
if __name__ == '__main__':
    main()
```