

Wine Quality Classification

Data Set: Wine Quality Dataset (White)

Author: Shuai Xu, email: sxu75374@usc.edu

Date: 5/8/2021

1. Abstract

In this project, I'm going to try different data pre-processing methods to process the data, includes Standardization/Normalization, dataset balance, feature engineering. Also, based on the pre-processed dataset, I will use six different machine learning algorithms to do the classification for the wine quality dataset which modified from the dataset from UCI machine learning Repository. The dataset contains 11 features and 3 classes which is good, medium and bad wine quality. Then, I create a probability-based trivial system and a baseline system by default perceptron with standardized data. For the six classification algorithms I chose are Linear Discriminant Analysis, K-Nearest Neighbors, Support Vector Machine, Stochastic Gradient Descend Classifier, Random Forest and Multi-Layer Perceptron. Then, I do the model selection by the cross-validation to get the best model and do the farther test by the test set. The main performance measures for the classification results comparison are: Accuracy, Macro f1 score and Confusion Matrix. Base on the classification result, I do the comparison with the trivial system and baseline system and do the comparison between each of the algorithms to do the farther analysis. Based on that, I find the Feature space expansion could help improve the model performance for all the classifier , Standardization/Normalization could improve the learning except the RF and LDA and Dataset balance could help improve the performance except for KNN. The order of the performance measures result I get is:

Test accuracy: RF>MLP>KNN>LDA>SVM>SGD>Baseline system >trivial system

Macro f1 score: RF>LDA>MLP>SVM>KNN>SGD>Baseline system>Trivial system

2. Introduction

2.1. Problem Statement and Goals

I choose the Wine Quality Dataset (White), which the original dataset is from the UCI Machine Learning Repository^[1]. The data is from the real world and each of the features will decide the quality of the wine quality. But, in my experiment, part of the dataset has been pre-processed. The dataset includes 11 features (fixed acidity, volatile acidity, var1*, var2*, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates and var3*, where var1*, var2*, var3* are the three features that has been pre-processed from the alcohol, residual sugar and citric acid) and 3 classes (class 1 is good, class 2 is medium, and class 3 is bad quality).

The goal is that I use some pre-processing techniques on the wine training and testing set to process the data, then implement several classification algorithms by validation or cross-validation to find good classification models, and finally use the best models I get to classify the test set to see the model performance for the unknown by the accuracy, macro f1 score and confusion matrix.

3. Approach and Implementation

3.1. Dataset Usage

3.1.1. Data Usage in Pre-processing

Original Data information (totally 2961 datapoints):

- Original Training Data:
The original Training dataset has
Class1: 622 datapoints
Class 2: 1339 datapoints
Class 3: 1000 datapoints
- Original Test Data (totally 500 datapoints):
Support datapoints in Class 1: 109
Support datapoints in Class 2: 208
Support datapoints in Class 3: 183

For the oversample dataset I used in this experiment is to enlarge the Class1 and Class3 to the same number of Class2(1339), thus, the oversample dataset will has 4017 datapoints in total.

Each of the model follow the four steps for the data pre-processing:

For the Pre-processing part, I use the cross-validation to get the performance of the different Standardization or Normalization methods for all the algorithms include in my experiment. I use it to get an average performance of the Standardization or Normalization to see whether I need to do the Standardization or Normalization. The cross-validation is sequential loops in this part, because there is only one parameter (Standardization or Normalization).

Table 1: Data use in parameter selection and training and testing the model

	Training data	Testing data	Validation data
Trivial system	2961	500	592
Perceptron	2961	500	592
LDA	4017	500	803
KNN	2961	500	592
SVM	4017	500	803
SGD	4017	500	803
RF	4017	500	803
MLP	4017	500	803

Table 2: How to process the data in pre-processing steps

	Pre-processing procedure			
	Step 1	Step 2	Step 3	Step 4
	Data balance	Feature engineering	Dimensionality adjustment	Standardization/Normalization
Perceptron	Imbalance	None	None	StandardScaler() (Standardization)
LDA	Oversample	Feature expansion (degree=2)	None	None
KNN	Imbalance	Feature expansion (degree=2)	None	StandardScaler() (Standardization)
SVM	Oversample	Feature expansion (degree=2)	None	StandardScaler() (Standardization)
SGD	Oversample	Feature expansion (degree=2)	None	StandardScaler() (Standardization)
RF	Oversample	Feature expansion (degree=2)	None	StandardScaler() (Standardization)
MLP	Oversample	Feature expansion (degree=2)	None	StandardScaler() (Standardization)

For the test set, it will be used when each of the step 2 and step 4 when we pre-process the training data. In each of the step 2, after use the StandardScaler() to standardize the training data, we also need to use the same scale to project the test set data on the training set (standardize the test set by the statistics of training set). Also, for the step 4, after we expand the feature space of the training set, we need to do the same to the testing set in order to make the feature of the test set has the same dimension with the training set.

Totally, in pre-processing section, test set was totally used for 12 times, includes 6 times feature expansion, 6 times standardization.

3.1.2. Data Usage in Model Selection

For the model selection, the Cross-validation I used for all the validation set I chose is the 1/5 number of the training set. Thus, the ratio of the validation training and validation set used for cross-validation is 4:1, where **the number**

of folds for cross-validation is 5 and the number of the same parameters in the cross-validation will be run 5 times. I use the GridSearchCV() from sklearn.model_selection library, this function could make the cross-validation by input the estimator and parameter tables to do the CV and it will return to use the result of CV and the best performance model. The fold number I choose 5 and use the validation set to test the model performance. The validation set is getting from the original training set. Cross-validation could easily help get the best model parameter. The exact number of datapoint I used in the cross-validation show in the Table 2.

To be more specific:

- LDA I only fix one parameter 'solver' by the cross-validation. Thus, the cross-validation will be sequential loops, only need to fix one parameter.
- KNN I only fix one parameter 'n_neighbors' by cross-validation. CV only need to take the sequential loops.
- SVM I fix two parameters by the cross-validation, 'C' and 'gamma'. In this case, cross-validation will try all the combination of the each value of C and gamma and return me the combination with the highest validation accuracy. Thus, the cross-validation will be the nested loops.
- SGD I also fix two parameters by the cross-validation, 'tol' and 'alpha'. Cross-validation returns me the best combination of 'tol' and 'alpha' with the highest validation accuracy. The cross-validation will be nested loops.
- Random Forest I also try the combination of the two parameters: 'max_depth' and 'n_estimators'. The cross-validation is nested loops to try every combination of those two parameters.
- MLP I try the combination of three different parameters: 'hidden_layer_sizes', 'alpha' and 'learning_rate_init' by the cross-validation. The loop will be the nested loops because CV needs to try all the combination of those three parameters

In model selection part, I don't do anything with the testset, except for testing the test accuracy and final macro f1 score. The model must be trained by the training set, the test set must not involve in this part in order to test the real model performance.

Totally, in model selection section, test set was totally used for 6 times, includes 6 times testing the best model which get from the cross-validation.

3.2. Preprocessing

3.2.1. Standardization and Normalization

3.2.1.1. Introduction and Comparison

Usually, the dataset will contain lots of different features. There will be a problem that the range and scale of these feature can be quite different. If we use the original data, those features with large range and scale will be given more weight. By that, we could use standardization method for the dataset to solve it and improve the performance.

For the pre-processing method I use the Standardization and Normalization to see the difference. For the Standardization, I use: `StandardScaler()`. For the normalization, I use the `MinMaxScaler()` and `MaxAbsScaler()` to see the difference, all of those three function are from the `sklearn.preprocessing` library.

- `StandardScaler()` is going to ignore the distribution and remove the mean of the data and divided by the standard deviation to center and rescale the data. The result will has 0 mean and unit variance. The `StandardScaler()` formula shows below:

$$x_{new} = \frac{x - \mu}{\sigma}$$

- `MinMaxScaler()`, it's a linear transformation to rescale the data based on the max and min value of the feature. The rescaled data will in the range of 0 to 1, is a normalization method. This method usually uses in the case of concentrated dataset. Because when the dataset is sparse, the standardized result will be quite unstable, and the result will be influenced by this. The formula shows below:

$$x_{new} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- `MaxAbsScaler()`, it will rescale the data to the range of -1 to 1, is a normalization method. This standardization method will keep the sparsity of the original feature and keep the original data distribution., because each feature divided by the max value of its own. The formula shows below:

$$x_{new} = \frac{x}{\max(x)}$$

The comparison based on all the model I tried in this experiment to see the standardization and normalization result. **All the comparison based on the original dataset without oversample/undersample in this Table 3 below.** The score used to do the comparison is the validation accuracy. All the validation set is from the original training data, the cross-validation dataset ratio is training:validation = 4:1. I use the StratifiedKFold() and train_test_split() function from the sklearn.model_selection library to do the cross-validation by myself, the number of fold I choose 5. The comparison result shows below, bold shows the better:

Table 3: Comparison for the performance of different Standardization or Normalization methods

Mean Validation accuracy	Raw data	After StandardScaler() (Standardization)	After MinMaxScaler() (Normalization)	After MaxAbsScaler() (Normalization)
Default Perceptron	0.3584486	0.4826307	0.46057336	0.44715008
Default LDA	0.5701855	0.5761889	0.57477234	0.57760539
Default KNN	0.4359528	0.5513659	0.55419899	0.56134907
Default SVM	0.4804047	0.60161889	0.59305228	0.551500843
Default SGD	0.3931197	0.52613828	0.53720067	0.53193929
Default RandomForest	0.5898145	0.59480607	0.59264755	0.59359191
Default ANN(MLP)	0.5483305	0.59177066	0.59291737	0.57780777

Based on table 4 above, I come up to the final choice listed in Table 4 with analysis:

Table 4: my final choice of Standardization/normalization method

	Standardization/normalization method
Perceptron(baseline)	StandardScaler() (Standardization)
LDA	None
KNN	StandardScaler() (Standardization)
SVM	StandardScaler() (Standardization)
SGDClassifier	StandardScaler() (Standardization)
Random Forest	StandardScaler() (Standardization)
ANN(MLP)	StandardScaler() (Standardization)

For LDA, the algorithm has been treated by scale-invariant, the standardization or normalization method will not influence the result of the LDA. For the result we could see in Table 4 that all the validation result is

almost same, the small difference is caused by the random chose validation set.

Also, for Random Forest, it's based on the each of the node to do the feature split. Each of the decision tree will be independent with other decision trees and each decision is based on only one feature, thus, will not be influenced by the other different features. Rescaling the feature in the short distance will not be quite important.

KNN is learning and do the classification based on the distance, thus, it's important to do the standardization to rescale the distance between each of the datapoint, especially for those features with a wide range of difference. We could use standardization to remove the impact of different distance scale. We could find that Standardization and Normalization could help a lot to compare with the original data. Although MaxAbsScaler() has the better result than StandardScaler(), I finally choose to use the StandardScaler(), because StandardScaler() will reduce the impact of outlier by 0 mean and unit variance and could help speed up the convergence.

Also, for SVM, it's the algorithm based on the computation on distance, thus, use the Standardization to rescale the data will be important. I use the Standardization to process the dataset, which could normal the dataset to the 0 mean and unit variance and help speed up the convergence and rescale the dataset to improve the classifier's performance.

For SGD and MLP, they are based error optimization by gradient descend, which will be influenced by the scale of each of the features. Standardization could help rescale the feature to a small range, which could make the optimization smoothly and could help accelerate the convergence and improve the performance.

3.2.2. Dataset Balance

For the imbalance dataset, usually some of the class will occupy a domain percentage of the whole dataset, this will cause the prediction of the data become unstable and even is a fake score. But this will cause by the dataset that has a large difference between classes, some small difference in certain extent is a normal phenomenon, especially in the real-world problem where the impact will not cause significant wrong or fake result.

There are two main method to do the rebalance, we called resample. The first one is undersample and the other one is oversample.

- Undersample is to remove the redundant datapoints, like the repeatedly datapoints, it compresses the number of datapoints to a certain selected number.
- Oversample is to randomly compensate the class that has small number of datapoints by creating new the datapoints with similar

feature or randomly choosing from the same class and add some noise to make it different.

In my experiment, **I choose to use the original dataset or using the oversample method**, because our dataset is not quite large, and the information of each datapoint is important, especially for the supervised machine learning method, such as random forest. The oversample method I choose to use is SMOTE() which is from the library `imblearn.over_sampling`, that could help oversample the training dataset by compensating each of the class by create some similar datapoints in the same class and add some noise to them.

For different model and algorithm, I list all the different method I choose for each of the model include in my experiment based on the comparison of the data balance by the test accuracy and macro f1 score. **All the comparison based on the original dataset without any modification, bold shows the better:**

Table 5: comparison the performance of Imbalance data and balance data

	Imbalance data		Balance data	
	Test accuracy	Macro f1	Test accuracy	Macro f1
LDA	0.56	0.54727	0.564	0.56091
KNN	0.422	0.40719	0.394	0.39479
SVM	0.436	0.30877	0.398	0.34507
SGD	0.348	0.32911	0.474	0.39445
RF	0.602	0.57997	0.612	0.60852
ANN(MLP)	0.488	0.43797	0.604	0.60091

Based on the comparison experiment above, the table below concludes the comparison result I use in the following sections:

Table 6: my final choice of data balance method

	Balance method
Perceptron (baseline)	Imbalance
LDA	Oversample to 1339/class
KNN	Imbalance
SVM	Oversample to 1339/class
SGD	Oversample to 1339/class
Random Forest	Oversample to 1339/class
ANN(MLP)	Oversample to 1339/class

(SVM I choose to balance the dataset in order to get a better macro f1 score but not a good test accuracy)

3.3. Feature engineering

For the feature engineering, I use the expanded feature space to expand the features. I use the function `PolynomialFeatures(degree=k)` from the `sklearn.preprocessing` library, to expand the feature space, where the k is the maximum degree of the new expanded feature space. For the choose of the degree k , I use the cross-validation to find the good degree value. The classifier I choose to do the cross validation is Linear Discriminant Analysis (LDA). This is because LDA will not influenced by the standardization^[2]. The LDA calculate the discriminant hyperplane by the datapoints, and standardization is a type of rescaling, which means will rescale the axis of the datapoints and will not influence the LDA classification result. Thus, I use LDA to test the effect of feature engineering (feature space expansion) to remove the influence caused by the standardization. The result shows below:

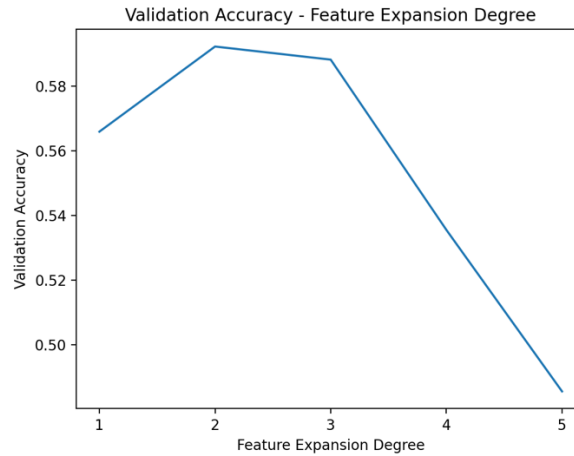


Figure 1: validation accuracy-feature expansion degree

When degree=1, which means the original features. We could find that when degree=2, the validation accuracy is the best, and when the degree is greater than 2, validation accuracy become lower, and the model begins overfitting. Thus, I use `PolynomialFeatures(degree=2)` to do the feature expansion as degree=2.

3.4. Feature dimensionality adjustment (if applicable)

For the dimension adjustment, the most common way is the PCA algorithm. But in the dataset in my experiment, I don't apply PCA on it, because all the features are important more or less. I use the `RandomForestClassifier()` from the `sklearn.ensemble` library, to output the feature importance score by using: `RandomForestClassifier().feature_importances_`, which could show the feature importance score. I test the feature importance based on both the original dataset and also the expansion feature space in section 3.3 to see the

results, I use the `pyplot.bar()` to draw the bar plot which from the `matplotlib.pyplot` library.

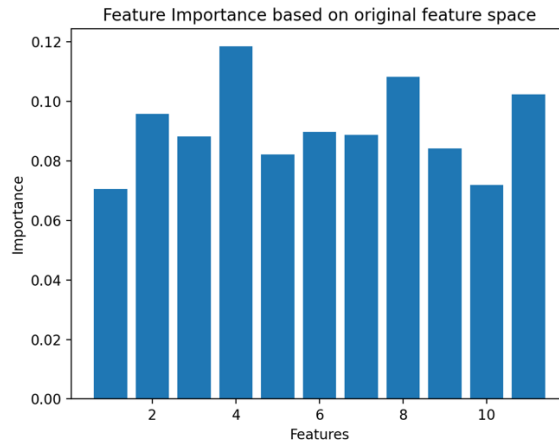


Figure 2: feature importance based on original feature space

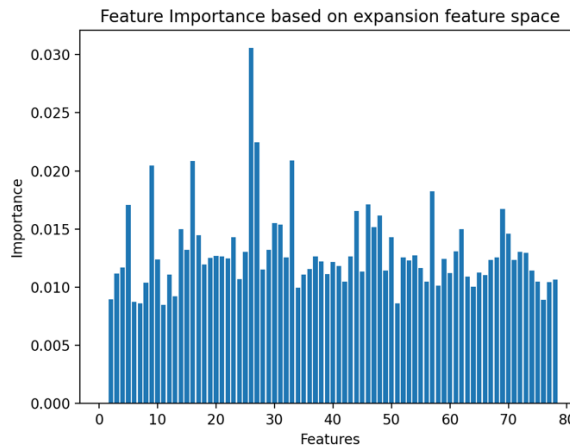


Figure 3: feature importance based on expansion feature space

From the results, we could find that all of the feature has very close importance score, especially for the original feature space. In expansion feature space, although there are six to seven features are more important than the others, the others similar important features are much more. Thus, if we use the PCA or `SelectKBest()` function which use to select the feature and reduce the dimension, lots of important information will be discarded, which will make the prediction result become worse, so that I don't implement feature dimension adjustment methods in my experiment.

3.5. Training, classification or regression, and model selection

3.5.1. Model 1: Linear Discriminant Analysis

3.5.1.1. Model and Algorithm

LDA is a supervised dimension reduction method, we could also use this to do the classification. LDA projects the data into low dimension hyperplane to make each of the classes has the largest inter-class distance, and the minimum distance within the same class, which means we find the direction of projection with the largest mean difference between different classes and the smallest variance of each of the class to gather the datapoints belong to the same class. Then based on this, we could use the same direction of projection to project the test data to the same hyperplane to see they are closer to which of the classes as the prediction result and to compare with the ground truth to calculate the accuracy. I use the `LinearDiscriminantAnalysis()` as my model, which from the `sklearn.discriminant_analysis` library.

3.5.1.2. Parameters

For the parameters, I do the Cross-validation for the parameter solver: 'svd' and 'lsqr' to see the difference. Singular Value Decomposition and Least Square are two methods to solve the projection result, SVD is good for large dataset. From the comparison result, I find that the result from the solver='svd' give me the best result. Thus, I finally use the `LinearDiscriminantAnalysis(priors=[622/2961, 1339/2961, 1000/2961], tol=0.0001, solver='svd')` as my final choice. I set the priors of the trainset to control the probability. The cross-validation I choose fold=5, training: validation = 4:1 to do the comparison.

3.5.1.3. Degree of Freedoms

For the original dataset, the d.o.f. is $C(D+1)$ which is $3*(11+1)=36$, but after I do the feature expansion as degree=2, d.o.f. is $C(D'+1)=3*(\frac{1}{2}*(11^2 + 3*11)+1) = 234$. The rule of thumb is $N_c > (3 \sim 10) * \text{d.o.f.} = 2340$. We have 2961 training datapoints which are quite good.

3.5.1.4. LDA results

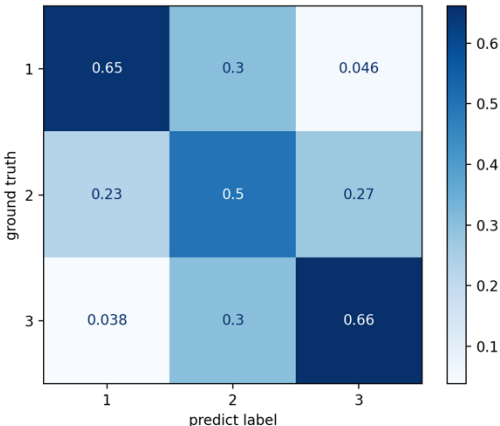
The input for training the best LDA model, I firstly balance the dataset by over-sampling the class 1(622) and class 3(1000) to the same number of class 2(1339) to do the farther classification. Because the domain of one of the classes will make the LDA result become worse, which shows in section 3.2.2. Then I do the polynomial feature expansion (degree=2) which shows is better than the original data in the section 3.3. The comparison result shows below:

Table 7: LDA cross-validation results

	Solver = 'svd'	Solver = 'lsqr'
Cross-validation mean accuracy	0.62012416,	0.33333333
Cross-validation mean std	0.02380945	0.00045473
Training accuracy	0.64177247	0.31006417

We could see that the validation accuracy for the 'svd' is better than using the least square. Then, I use the final best model `LinearDiscriminantAnalysis (priors=[622/2961, 1339/2961, 1000/2961], solver='svd')` to test the test set. The result I get is from the function `classification_report()` which could show me both the test accuracy and the macro f1 score. This function is from the `sklearn.metrics` library.

Table 8: Performance of the final LDA best system

Test accuracy	0.592																
Macro f1	0.59601																
Confusion matrix	<div><p>confusion matrix for Model 1: LDA final testset</p><table><tr><th>ground truth \ predict label</th><th>1</th><th>2</th><th>3</th></tr><tr><th>1</th><td>0.65</td><td>0.3</td><td>0.046</td></tr><tr><th>2</th><td>0.23</td><td>0.5</td><td>0.27</td></tr><tr><th>3</th><td>0.038</td><td>0.3</td><td>0.66</td></tr></table></div>	ground truth \ predict label	1	2	3	1	0.65	0.3	0.046	2	0.23	0.5	0.27	3	0.038	0.3	0.66
ground truth \ predict label	1	2	3														
1	0.65	0.3	0.046														
2	0.23	0.5	0.27														
3	0.038	0.3	0.66														

I use the `confusion_matrix()` function to calculate the confusion matrix and `plot_confusion_matrix()` function to visualize the confusion matrix, both of the function from the `sklearn.metrics` library. After applying the best model, we could find that the test accuracy is 0.592 and macro f1 is 0.59373. And we could find the confusion matrix class 1 and class 3 is better than the class 2.

3.5.2. Model 2: K Nearest Neighbors

3.5.2.1. Model and Algorithm

K Nearest Neighbors find the nearest k datapoints of the target datapoint, and vote to decide the label of the target datapoint by the k nearest neighbors, the

target datapoint will be assigned by the most frequent label in the K nearest datapoints. For example, if the k nearest neighbors has more than $\frac{1}{2}$ datapoints which belongs to class 1, the target datapoint will be predicted as class 1. The nearest datapoint is defined by the shortest distance, it's non-parametric classifier. In my experiment, I use the `KNeighborsClassifier()` as the function to do the classification, which from the `sklearn.neighbors` library.

3.5.2.2. Parameters

The most important parameter in KNN algorithm is the number of nearest neighbors: K. We could well-tune the number of K to improve the performance of the KNN algorithm. If K is too large, the far datapoints will be included as nearest neighbors, which will cause the high deviation of the class, the K nearest will not be representative and finally lower the accuracy and cause the underfit. If K is too small, such as 1 nearest neighbor, the system will perform bad when the datapoints from the different classes has a very short distance between each other and finally cause the overfit.

Thus, I use the cross-validation to try different `n_neighbors` (K) to see the result. The numbers of K I tried are [1,2,5,8,10,15,20,50,60,65,70,75,80,90], where I take 14 different numbers from a small number 1 to a large number 90. The cross-validation result gives me that when `n_neighbors=60` is the best. The cross-validation I choose `fold=5`, training: validation = 4:1 to do the comparison:

Table 9: KNN cross-validation results

validation	K=5 (default)	K=20	K=60	K=70	K=90
Mean accuracy	0.50354018	0.56028269	0.58425607	0.57277471	0.57952577
Mean std	0.01612732	0.02177122	0.01138281	0.01652899	0.01731269
Training accuracy	1.0	1.0	1.0	1.0	1.0

I choose some of the cross-validation result to see the difference. We could see that when K is too large or too small, both of the result will be not good. Thus, I choose `KNeighborsClassifier(n_neighbors=60, weights='distance')` as my final choice which is the max score the CV gives to me. Also, we could see all the training accuracy is 1.0

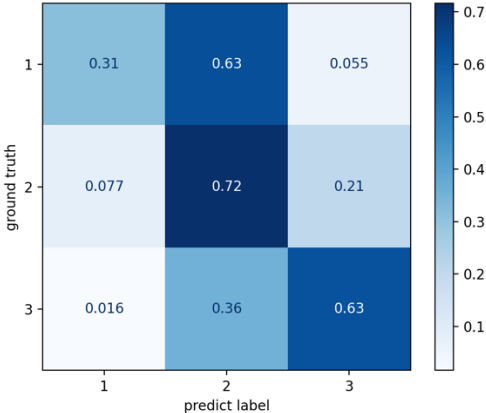
3.5.2.3. KNN results

The input for training the KNN best model, I choose to use the imbalanced expanded feature space with `degree=2` and then do the standardization by

StandardScaler(). The reason shows in the previous section 3.2.1.1, 3.2.2 and 3.3. Based on the best parameter cross-validation gives to me: I use KNeighborsClassifier (n_neighbors=60, weights='distance') to test the test set to see the final result for the best model I get.

The result shows below:

Table 10: Performance of the final KNN best system

Test accuracy	0.596																
Macro fl	0.56317																
Confusion matrix	<div><p>confusion matrix for Model 2: KNN final testset</p><table><tr><th>ground truth \ predict label</th><th>1</th><th>2</th><th>3</th></tr><tr><th>1</th><td>0.31</td><td>0.63</td><td>0.055</td></tr><tr><th>2</th><td>0.077</td><td>0.72</td><td>0.21</td></tr><tr><th>3</th><td>0.016</td><td>0.36</td><td>0.63</td></tr></table></div>	ground truth \ predict label	1	2	3	1	0.31	0.63	0.055	2	0.077	0.72	0.21	3	0.016	0.36	0.63
ground truth \ predict label	1	2	3														
1	0.31	0.63	0.055														
2	0.077	0.72	0.21														
3	0.016	0.36	0.63														

We could see from the confusion matrix, most of class 1 is confused with class 2, which means lots of datapoints from class 1 is surrounded by datapoints from class 2 and hard to discriminant.

3.5.3. Model 3: Support Vector Machine

3.5.3.1. Model and Algorithm

SVM is designed to do the classification for the two classes classification problem, especially is the optimization algorithm of solving convex quadratic problem. The basic idea of SVM is to find the hyperplane that could do the classification with the minimum error or correctly do the classification and then find hyperplane that could make the interval which between the hyperplane supported by the support vectors largest. We could see the image ^[3]below:

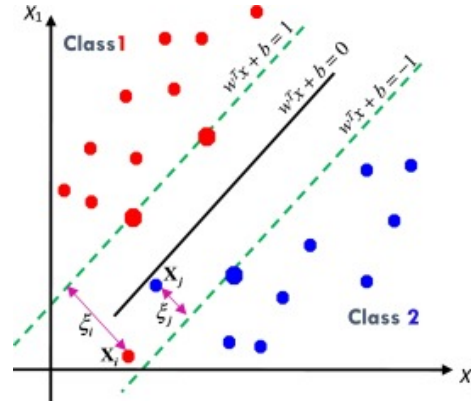


Figure 4: principal of SVM

We need to make the interval ζ largest by separate the hyperplanes supported by the support vectors and correctly classify the datapoints. In our case is the multiclass problem ($C=3$), thus, we need to use the OneVsRest method to do the multiclass classification. In my experiment, I use the function `SVC()` as my SVM model from the `sklearn.svm` library.

3.5.3.2. Parameters

There are three main parameters in the SVM algorithm, C, gamma and kernel function. For the Kernel function, I simply set to the 'rbf' to deal with the non-linear problem which are much useful in the real-world problem. Because the rbf kernel could map the data to the new log space by the exponential curve. Thus, I use the Cross-Validation to test different combination of C and gamma to see the comparison results. The numbers of C and gamma I tried 15 for each, I use the `np.logspace(-1, 3, num=15)` to generate the 15 different C value form the range of 10^{-1} to 10^3 and use the `np.logspace(-3, 1, num=15)` to generate the 15 different gamma value from of 10^{-3} to 10^1 , where gamma is the coefficient of the RBF kernel function and C is the regulation of the l2 penalty. A large penalty will increase the accuracy but may cause the overfitting problem.

After doing the cross-validation, finally, the best result I get from it is: $C = 268.26957952797244$ and $\text{gamma} = 0.0071968567300115215$.

Because I use the `logspace` to generate C and gamma, I decide to use the int number to instead and select around the 268, because the gap between two of the numbers is quite large, thus I decide to use $C=150$ or 200 . **Finally, I use `SVC(C=150, gamma=0.001, kernel='rbf', decision_function_shape='ovr')` as my final choice.** The cross-validation I choose `fold=5`, training: validation = 4:1 to do the comparison. Thus, it will be $15 \times 15 \times 5$ times cross-validation, the results are too much to show them. The mean validation accuracy of the best model is and the according mean validation std is 0.01295084.

3.5.3.3. Degree of Freedom

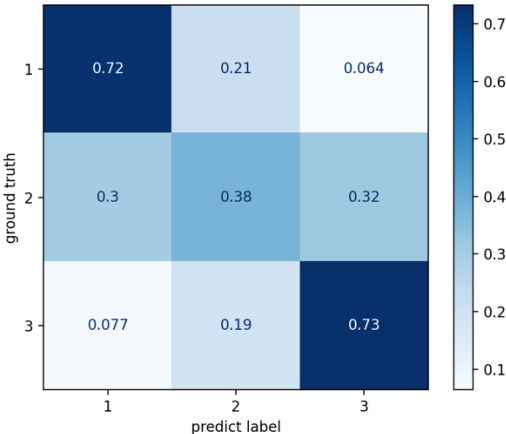
Because SVM for multiclass classification should use the OVR or OVO method, I use OVR in this problem in order to reduce the complexity. Based on OVR, the OVR SVM classifier should have a d.o.f. of $C(D'+1)$ for the expansion feature space, which is $3 * (\frac{1}{2} * (11^2 + 3 * 11) + 1) = 234$. I do the oversample for the OVR SVM classifier in order to more easily get the decision hyperplane. The resampled data has $1339 * 3 = 4017$ datapoints, which is greater than 10 times of the d.o.f..

3.5.3.4. SVM results

The input for training the OVR SVM best model, I choose to use the oversampled dataset (all of the three classes has 1339 datapoints) which could help define the hyperplane with more possible support vectors. The comparison shows in the table in section 3.2.2, the balanced data will help get a much better macro f1 score. Then, I do the feature space expansion as degree=2 to make the decision accuracy more precisely and better, help the machine learn more information from the dataset. Then I use the StandardScaler() to standardize the data to improve the performance. Based on the CV result in 3.5.3.2, I retrain it by the training set to see prediction result for the test set. The model I use is: `SVC(C=150, gamma=0.001, kernel='rbf', decision_function_shape='ovr')`

The result shows below:

Table 11: Performance of the final OVR SVM best system

Test accuracy	0.584																
Macro f1	0.58046																
Confusion matrix	<div>confusion matrix for Model 3: SVM final testset</div>  <table><thead><tr><th></th><th>1</th><th>2</th><th>3</th></tr></thead><tbody><tr><th>1</th><td>0.72</td><td>0.21</td><td>0.064</td></tr><tr><th>2</th><td>0.3</td><td>0.38</td><td>0.32</td></tr><tr><th>3</th><td>0.077</td><td>0.19</td><td>0.73</td></tr></tbody></table>		1	2	3	1	0.72	0.21	0.064	2	0.3	0.38	0.32	3	0.077	0.19	0.73
	1	2	3														
1	0.72	0.21	0.064														
2	0.3	0.38	0.32														
3	0.077	0.19	0.73														

We could find by the confusion matrix that the class 2 is confused with both of the class 1 and class 3, but class 1 and class 3 performs well has an over 0.7 correct rate.

3.5.4. Model 4: SGD Classifier

3.5.4.1. Model and Algorithm

Stochastic Gradient Descent is a simple method to do the classification based on the SVM or Logistic Regression, which are the convex loss functions. SGD find the direction of the fastest gradient descend to solve the problem. This method has a very fast convergence speed when the dataset is standardized and could deal with the very large dataset with large number of features. SGD is very efficient and easy to implement but we need to use the standardization or normalization to normal the dataset Because we need to use the features to calculate the gradient, if the range of the features are different, the gradient descend result will be hard to convergence. Thus, we usually use the standardized rescaled data to make the feature become closer and that will accelerate the convergence. In my experiment, I use SGDClassifier() function as the SGD Classifier model and implement to solve the problem, which form the sklearn.linear_model.

3.5.4.2. Parameters

In SGDClassifier(), there are lots of parameters in it. Basically, I simply set the 'max_iter' to the 10^8 to ensure the convergence of the SGD. The loss parameter I use the 'hinge' which is linear svm. Then, for the 'alpha' parameter, this is the regulation term to help get good result and 'tol' parameter is the term that use to set the halt condition-if the current loss is greater than the (best loss - tol), the system will stop training. Thus, I use the cross-validation to find the good 'alpha' and 'tol' parameters. The range I set for 'alpha' is [0.0001,0.0002, 0.0003, 0.0004, 0.0005, 0.001, 0.05,0.1,1], which has 9 different numbers and the range of 'tol' I set as [0.00000001, 0.0000005, 0.000001, 0.00005, 0.0001, 0.001, 0.005], where has 7 different numbers. The cross-validation I use the number of folds is 5, the training set: validation set is 4:1. After the cross-validation for each of the combination, the best paramters I get I {'alpha': 0.001, 'tol': 1e-06}. There are 9*7*5 times of CV, thus, I select some of the CV result list in the table below:

Table 12: SGD cross-validation results

CV result	'alpha': 0.0001, 'tol': 1e-08	'alpha': 0.001, 'tol': 1e-06	'alpha': 1, 'tol': 0.005
Validation mean accuracy	0.5241443	0.55792637	0.51233308

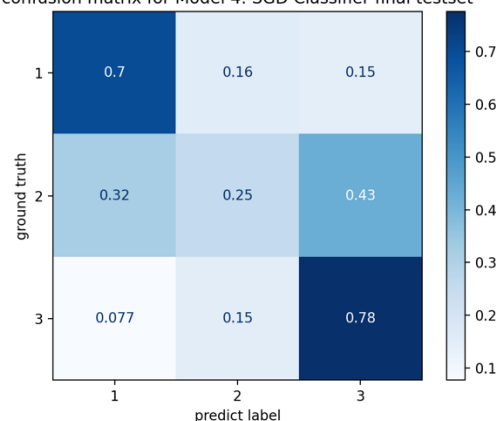
Validation mean std	0.03321703	0.01379969	0.02106647
------------------------	------------	-------------------	------------

Thus, finally I choose the best parameter SGDClassifier(alpha=0.001, tol=1e-06, max_iter=10⁸, loss='hinge') to test the test dataset.

3.5.4.3. SGD results

The input for training the SGDClassifier best model, I choose to use the oversampled dataset (all of the three classes have 1339 datapoints for each). The comparison shows in the table in section 3.2.2, the balanced data will help get a much better result. Then, I do the feature space expansion as degree=2 to help the machine learn more information from the dataset. Finally, I use the Standardization, because SGD is quite sensitive to the Standardization and will help improve the performance and convergence speed. The result shows below:

Table 13: Performance of the final SGD best system

Test accuracy	0.552																
Macro f1	0.54007																
Confusion matrix	<div>confusion matrix for Model 4: SGD Classifier final testset</div>  <table><tr><th>ground truth \ predict label</th><th>1</th><th>2</th><th>3</th></tr><tr><th>1</th><td>0.7</td><td>0.16</td><td>0.15</td></tr><tr><th>2</th><td>0.32</td><td>0.25</td><td>0.43</td></tr><tr><th>3</th><td>0.077</td><td>0.15</td><td>0.78</td></tr></table>	ground truth \ predict label	1	2	3	1	0.7	0.16	0.15	2	0.32	0.25	0.43	3	0.077	0.15	0.78
ground truth \ predict label	1	2	3														
1	0.7	0.16	0.15														
2	0.32	0.25	0.43														
3	0.077	0.15	0.78														

We could see that the accuracy of class 1 and class 3 is good, but most of class 2 is confused with class 1 and class 3.

3.5.5. Model 5: Random Forest

3.5.5.1. Model and Algorithm

Random Forest is a Bootstrap Aggregation algorithm, and the Random Forest is formed by lots of Decision Trees which is a supervised learning algorithm

based on some intuitive if, then, else decision. One of the decision tree doesn't has the correlation with the other decision tree in the random forest. The input dataset will get the prediction by each of the decision tree and finally the prediction of the random forest will be the most frequency result gave by all of the decision trees in the forest. The training of decision tree will base on the random choose samples from the dataset with replacement. When the decision reaches the node, it needs to be split, that when each sample has N attributes, they will be randomly selected n from N where $n \ll N$. Then decision tree uses different criterion strategy to select one attribute for this node to be the split attributes from the n attributes. Then, repeat this step to create more branches until the attributes randomly chose is same to the last node, which means decision tree reach the leaf, then the system will halt. I implement the Random Forest use the RandomForestClassifier() in my experiment, which from the sklearn.ensemble library.

3.5.5.2. Parameters

There are two main parameters in the RandomForestClassifier(), the first one is 'n_estimator', which is the number of the trees in the forest and the other is max_depth, which is the max depth of the tree. The criterion I decide to use the 'entropy' to make desicion. For the decide of the number of the trees and the max depth, I use the cross-validation to find the best pair of them. 'max_depth' I tried 10 settings in the range from 5 to 25 with step=2, n_estimators I tried 15 settings in the range from 15 to 101 with step=10.

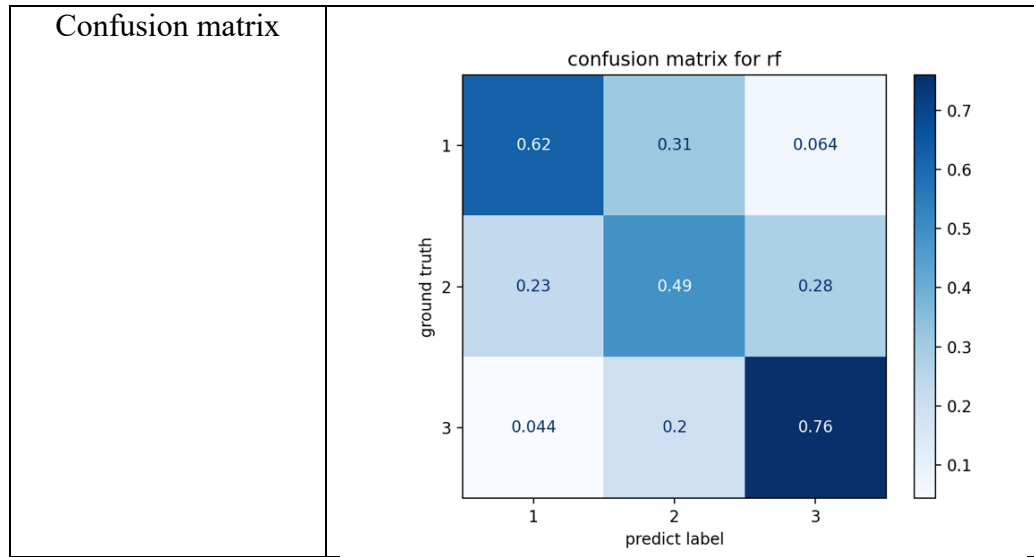
After doing the cross-validation, the best result I get is from: max_depth=150, n_estimator=21, the best mean validation accuracy is 0.67738966 and the according mean validation std is 0.04078865. Finally, I choose to use RandomForestClassifier (n_estimators=150, max_depth=,21 random_state=0) as my final model to test the testing set.

3.5.5.3. Random Forest results

For the input of the RandomForestClassifier() best model, I choose to use the oversampled dataset (all of the three classes have 1339 datapoints for each). The comparison shows in the table in section 3.2.2, the balanced data will help get a much better macro f1 result. Then, I do the feature space expansion as degree=2 to help the machine learn more information from the dataset. But I still use the standardized data to reduce the computation.

Table 14: Performance of the final RF best system

Test accuracy	0.618
Macro f1	0.61296



We could see that the accuracy of class 3 is the best and half of class 2 is confused with class 1 and class 3. And class 1 is also easy to confuse with class 2.

3.5.6. Model 6: Multi-Layer Perceptron

3.5.6.1. Model and Algorithm

Multi-Layer Perceptron is a kind of Artificial Neural Network, each of the node of the layer is fully connected to the nodes of next layer. Basically, the structure of the network includes input layer, hidden layer and output layers. The number of hidden layers could be one or more. Nodes include the non-linear activation function in it. Supervised Back-Propagation used as the method to train the model by minimum the error in each of the iteration in the BP network, use the error to fix the weight to get the final best result. The computation of each of the nodes is based on the summation of the inner product of the input data and the according weight value. In my experiment, I implement the Multi-Layer Perceptron by the function `MLPClassifier()`, which from the `sklearn.neural_network` library.

3.5.6.2. Parameters

There are three main parameters in the `MLPClassifier()`. The first one is 'hidden_layer_sizes', which is the number of the hidden layers in the neural network, 'alpha', which is regulation term of the regression and 'learning_rate_init', which set the learning rate. For the fixed term, I set the momentum as 0.95 to increase the convergence speed, where momentum is similar to the momentum in the physics. Max_iter is set to 100000 to make it enough to convergence. The criterion function I set to 'adam', which could fix the learning rate by itself on the different weight value for the gradient

descend. For the 'hidden_layer_sizes', set it to [10, 30, 50, 100, 150, 200,300,500], 8 numbers. 'alpha' I set to two numbers: [0.0001, 0.001] to make it small to get a large penalty for wrong classification. Then, I set the learning_rate_init to three numbers: [0.00001,0.0001, 0.001], select a smaller one to improve the accuracy.

Based on those, I run the cross-validation to see the result of those different combinations. Finally I get the best model by the CV is:

MLPClassifier(hidden_layer_sizes=500,solver='adam',random_state=1, max_iter=100000, momentum=0.95, learning_rate_init=0.00001), where the validation accuracy is 0.6039 and std is 0.0232.

3.5.6.3. MLP results

The input for the training of Multi-Layer Perceptron Classifier, I choose to use the oversampled dataset (all of the three classes have 1339 datapoints for each). And balanced data will help get better result. Then, I do the feature space expansion as degree=2 to help the machine learn more information from the dataset. Because standardized data could improve the performance of MLP, I use the StandardScaler() to standardize the dataset.

Table 15: Performance of the final MLP best system

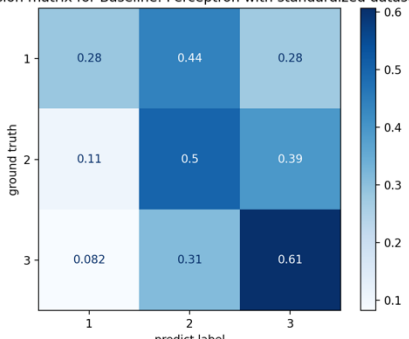
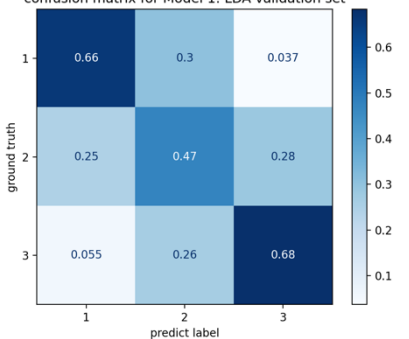
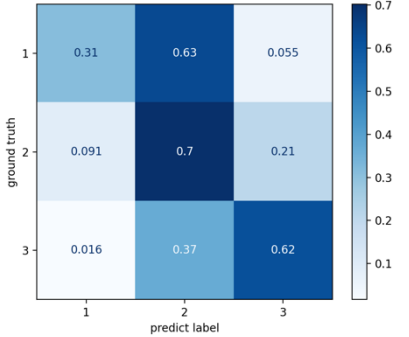
Test accuracy	0.6040																
Macro f1	0.5911																
Confusion matrix	<div>confusion matrix for Model 6: Multi-Layer Perceptron</div> <table><thead><tr><th></th><th>1</th><th>2</th><th>3</th></tr></thead><tbody><tr><th>1</th><td>0.71</td><td>0.22</td><td>0.073</td></tr><tr><th>2</th><td>0.26</td><td>0.4</td><td>0.34</td></tr><tr><th>3</th><td>0.055</td><td>0.2</td><td>0.74</td></tr></tbody></table>		1	2	3	1	0.71	0.22	0.073	2	0.26	0.4	0.34	3	0.055	0.2	0.74
	1	2	3														
1	0.71	0.22	0.073														
2	0.26	0.4	0.34														
3	0.055	0.2	0.74														

We could see that Class 1 and Class3 has a good result and Class 2 is confused with both.

4. Analysis: Comparison of Results, Interpretation

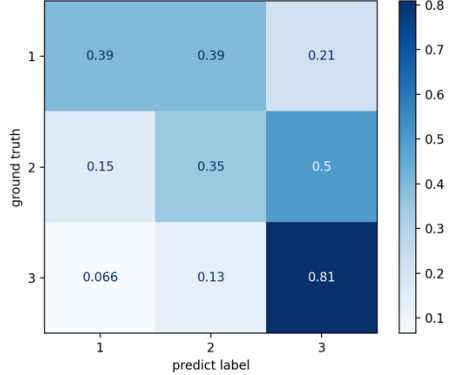
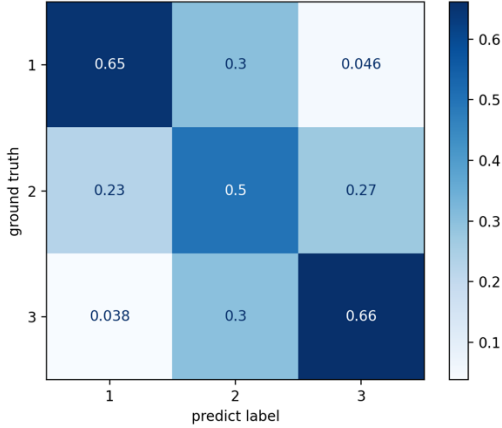
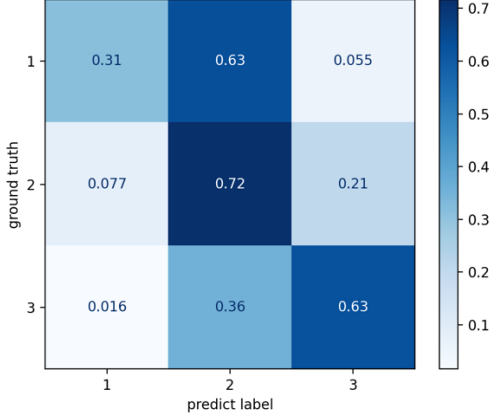
4.1. Comparison of Results

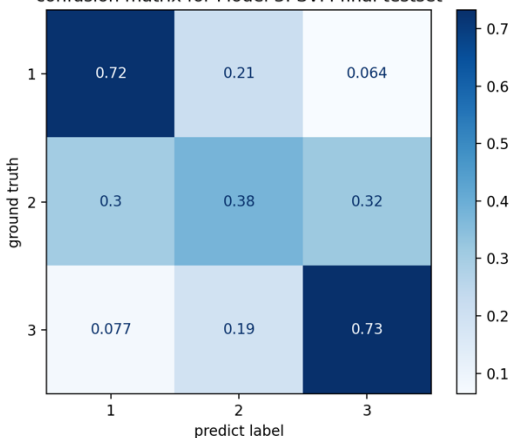
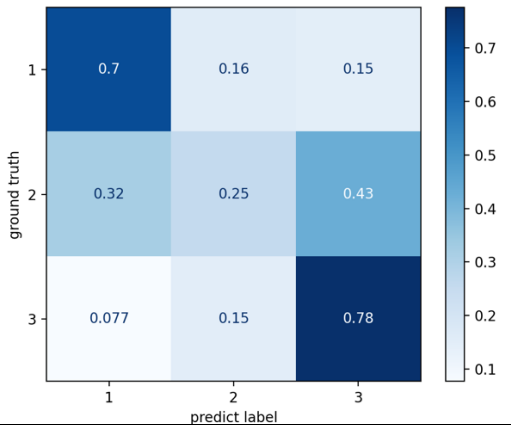
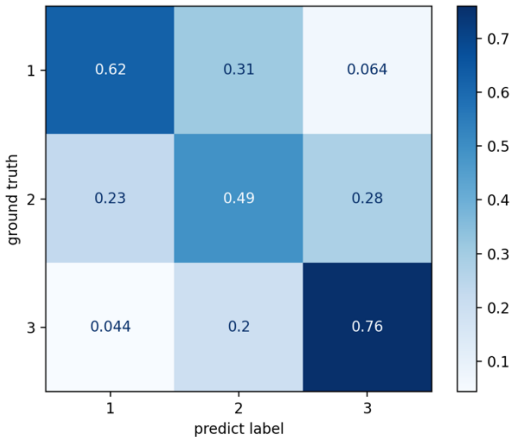
Table 16: Performance measures for Validation set

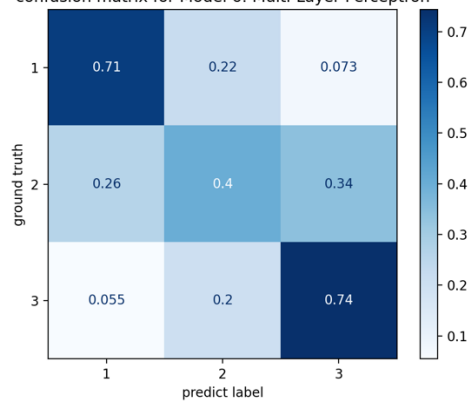
Validation set (digits=4)	accuracy	Marco fl	Test set Confusion Matrix for the best model
Trivial system	0.402	0.3755	[[0.24770642 0.52293578 0.2293578] [0.14423077 0.51442308 0.34134615] [0.24043716 0.39344262 0.36612022]]
Baseline	0.492	0.46529	<p>confusion matrix for Baseline: Perceptron with standardized dataset</p> 
LDA	0.59	0.59179	<p>confusion matrix for Model 1: LDA validation set</p> 
KNN	0.586	1.0	<p>confusion matrix for Model 2: KNN validation set</p> 

SVM	0.574	0.57054	<p>confusion matrix for Model 3: SVM validation set</p> <table border="1"> <thead> <tr> <th>ground truth \ predict label</th><th>1</th><th>2</th><th>3</th></tr> </thead> <tbody> <tr> <th>1</th><td>0.72</td><td>0.24</td><td>0.046</td></tr> <tr> <th>2</th><td>0.3</td><td>0.36</td><td>0.34</td></tr> <tr> <th>3</th><td>0.071</td><td>0.2</td><td>0.73</td></tr> </tbody> </table>	ground truth \ predict label	1	2	3	1	0.72	0.24	0.046	2	0.3	0.36	0.34	3	0.071	0.2	0.73
ground truth \ predict label	1	2	3																
1	0.72	0.24	0.046																
2	0.3	0.36	0.34																
3	0.071	0.2	0.73																
SGD	0.542	0.51403	<p>confusion matrix for Model 4: SGD Classifier validation set</p> <table border="1"> <thead> <tr> <th>ground truth \ predict label</th><th>1</th><th>2</th><th>3</th></tr> </thead> <tbody> <tr> <th>1</th><td>0.74</td><td>0.11</td><td>0.15</td></tr> <tr> <th>2</th><td>0.36</td><td>0.19</td><td>0.45</td></tr> <tr> <th>3</th><td>0.098</td><td>0.082</td><td>0.82</td></tr> </tbody> </table>	ground truth \ predict label	1	2	3	1	0.74	0.11	0.15	2	0.36	0.19	0.45	3	0.098	0.082	0.82
ground truth \ predict label	1	2	3																
1	0.74	0.11	0.15																
2	0.36	0.19	0.45																
3	0.098	0.082	0.82																
RF	0.592	0.58997	<p>confusion matrix for Random Forest Validation set</p> <table border="1"> <thead> <tr> <th>ground truth \ predict label</th><th>1</th><th>2</th><th>3</th></tr> </thead> <tbody> <tr> <th>1</th><td>0.61</td><td>0.33</td><td>0.055</td></tr> <tr> <th>2</th><td>0.24</td><td>0.47</td><td>0.29</td></tr> <tr> <th>3</th><td>0.044</td><td>0.24</td><td>0.72</td></tr> </tbody> </table>	ground truth \ predict label	1	2	3	1	0.61	0.33	0.055	2	0.24	0.47	0.29	3	0.044	0.24	0.72
ground truth \ predict label	1	2	3																
1	0.61	0.33	0.055																
2	0.24	0.47	0.29																
3	0.044	0.24	0.72																
MLP	0.57	0.56831	<p>confusion matrix for Model 6: Multi-Layer Perceptron validation set</p> <table border="1"> <thead> <tr> <th>ground truth \ predict label</th><th>1</th><th>2</th><th>3</th></tr> </thead> <tbody> <tr> <th>1</th><td>0.67</td><td>0.26</td><td>0.073</td></tr> <tr> <th>2</th><td>0.29</td><td>0.39</td><td>0.31</td></tr> <tr> <th>3</th><td>0.055</td><td>0.23</td><td>0.71</td></tr> </tbody> </table>	ground truth \ predict label	1	2	3	1	0.67	0.26	0.073	2	0.29	0.39	0.31	3	0.055	0.23	0.71
ground truth \ predict label	1	2	3																
1	0.67	0.26	0.073																
2	0.29	0.39	0.31																
3	0.055	0.23	0.71																

Table 16: Comparison Results for Test set

(digits=4)	Train accuracy for the best model	Test accuracy for the best model	Test set Macro f1 for the best model	Macro f1 on Vocareum	Test set Confusion Matrix for the best model
Trivial system	0.3506	0.324	0.3029	None	[[0.20183486 0.47706422 0.32110092] [0.20192308 0.41346154 0.38461538] [0.21857923 0.4863388 0.29508197]]
Baseline	0.4948	0.526	0.50042	0.5	<p>confusion matrix for Baseline: Perceptron with standardized dataset</p> 
LDA	0.6440	0.592	0.5960	0.596	<p>confusion matrix for Model 1: LDA final testset</p> 
KNN	1.0	0.5960	0.5632	0.563	<p>confusion matrix for Model 2: KNN final testset</p> 

SVM	0.6820	0.584	0.58046	0.580	<div>confusion matrix for Model 3: SVM final testset</div>  <table><tr><th>ground truth \ predict label</th><th>1</th><th>2</th><th>3</th></tr><tr><th>1</th><td>0.72</td><td>0.21</td><td>0.064</td></tr><tr><th>2</th><td>0.3</td><td>0.38</td><td>0.32</td></tr><tr><th>3</th><td>0.077</td><td>0.19</td><td>0.73</td></tr></table>	ground truth \ predict label	1	2	3	1	0.72	0.21	0.064	2	0.3	0.38	0.32	3	0.077	0.19	0.73
ground truth \ predict label	1	2	3																		
1	0.72	0.21	0.064																		
2	0.3	0.38	0.32																		
3	0.077	0.19	0.73																		
SGD	0.6196	0.5520	0.5401	0.54	<div>confusion matrix for Model 4: SGD Classifier final testset</div>  <table><tr><th>ground truth \ predict label</th><th>1</th><th>2</th><th>3</th></tr><tr><th>1</th><td>0.7</td><td>0.16</td><td>0.15</td></tr><tr><th>2</th><td>0.32</td><td>0.25</td><td>0.43</td></tr><tr><th>3</th><td>0.077</td><td>0.15</td><td>0.78</td></tr></table>	ground truth \ predict label	1	2	3	1	0.7	0.16	0.15	2	0.32	0.25	0.43	3	0.077	0.15	0.78
ground truth \ predict label	1	2	3																		
1	0.7	0.16	0.15																		
2	0.32	0.25	0.43																		
3	0.077	0.15	0.78																		
RF	0.9997	0.6180	0.6129	0.614	<div>confusion matrix for rf</div>  <table><tr><th>ground truth \ predict label</th><th>1</th><th>2</th><th>3</th></tr><tr><th>1</th><td>0.62</td><td>0.31</td><td>0.064</td></tr><tr><th>2</th><td>0.23</td><td>0.49</td><td>0.28</td></tr><tr><th>3</th><td>0.044</td><td>0.2</td><td>0.76</td></tr></table>	ground truth \ predict label	1	2	3	1	0.62	0.31	0.064	2	0.23	0.49	0.28	3	0.044	0.2	0.76
ground truth \ predict label	1	2	3																		
1	0.62	0.31	0.064																		
2	0.23	0.49	0.28																		
3	0.044	0.2	0.76																		

MLP	0.6805	0.6040	0.5911	0.591	<div><p>confusion matrix for Model 6: Multi-Layer Perceptron</p><table><caption>Confusion Matrix Data</caption><thead><tr><th>ground truth \ predict label</th><th>1</th><th>2</th><th>3</th></tr></thead><tbody><tr><th>1</th><td>0.71</td><td>0.22</td><td>0.073</td></tr><tr><th>2</th><td>0.26</td><td>0.4</td><td>0.34</td></tr><tr><th>3</th><td>0.055</td><td>0.2</td><td>0.74</td></tr></tbody></table></div>	ground truth \ predict label	1	2	3	1	0.71	0.22	0.073	2	0.26	0.4	0.34	3	0.055	0.2	0.74
ground truth \ predict label	1	2	3																		
1	0.71	0.22	0.073																		
2	0.26	0.4	0.34																		
3	0.055	0.2	0.74																		

The trivial system I designed will predict the output label by the probability, I use the `random.choice()` function to choose the three label 1,2,3 as the predict result, I set the random choose with replacement and will choose label 1 with probability=622/2961, label 2 with probability=1339/2961 and label 3 with probability=1000/2961. We could see that the test accuracy is around 0.3, which is not quite good. Also, the f1 score is only 0.3. Because this system does not support by any learning algorithms, just use the probability to predict. From the confusion matrix, we could find that are the entries of the matrix seems uniformly distribution, there don't have any good result on the diagonal.

The Baseline system we could see, it has a very good performance than the trivial system, especially after the standardization. The test accuracy is 0.526 and f1 score is 0.50042 which is significantly better than trivial system. For baseline, we just use the default linear Perceptron with the standardized data to do the classification. If we well-tune the Perceptron parameters, the performance will be better perhaps. For the confusion matrix, the class 3 has good result, but for the class 1 and class 2, both of them are worse than other algorithms. I think is because the Baseline system use the imbalance dataset.

For the LDA algorithm, the performance beyond my imagination. To compare with the trivial system and baseline, it performs better than both of them. The test accuracy and macro f1 is almost achieve 0.6. To compare with all the other algorithm, especially those advanced algorithms, like MLP and RF, the performance result seems quite similar and for those traditional algorithms, LDA performs better than those. For the reason why I think is because the dataset is high-dimensional and use the learning algorithm may cause the overfitting or underfitting problem easily. But for LDA, it projects the datapoints to the lower-dimensional space which may create more separable clusters to get a better classification result. Also, we could find in the confusion matrix that LDA has a good performance in Class2 than others except for KNN. I think it's because the LDA find the most separable direction of projection, which make the Class 2 separable and don't all mixed with other classes in the new lower dimension.

For KNN, it also has a better performance than the trivial system and the baseline system. I use the cross-validation carefully refine the model to get a better result. It finally get a test accuracy of 0.596 and f1 score of 0.5632. To compare with the other algorithm, KNN

performs better than the SGD and has a similar performance with SVM but is not better than the others. For the reasons, I think KNN deal better with the outliers than the SGD and it performs better for the sparse data distribution, which is good than SGD. But for the training time, SGD will be better than KNN, because KNN need to compute the distance with all of the datapoints, especially for a larger dataset. Knn is good for the multiclass classification, which is better than SVM, and don't have the training time. For itself, when the samples are not balanced, the prediction accuracy for the class that has small number of datapoints will be bad. Especially, when the sample size has very large difference, the sample from small size class will easily be dominated by large one and lead to be classified to the large class. Also, if there are some errors in the training data set, which are just around to the target datapoint, it will lead to the misclassification. We could see that the KNN training accuracy is easy to be 1, because the training data just to compare with the training data. For the confusion matrix, we could find that KNN is quite different with other algorithms, because the best classification is the class 2, which is the worst one in other algorithms. I think is because both of the datapoints from class 1 and class 3 is close to the class 2.

For SVM, it performs better than the trivial and baseline system, and also better than the SGD for both of the f1 and test accuracy but is not better than the other algorithms. For the reason, I think SVM is a very good method for the two-class classification problem, but for the multiclass problem, we need to use the OvR method and take the maximum of overlapping area to deal with the multiclass problem. It also quite time and space consuming. It performs better than SGD after I well-tune the C and gamma parameters. For the confusion matrix of SVM, it looks same with the SGD, the Class 1 and 3 has a very good result, but class 2 is not good. I think it's because the SVM is designed for 2 class classification, the OVR may not perform well on the medium class 2.

For the SGDClassifier, it performs better than the trivial system a lot and better than baseline system a little bit. Also, SGD performs worse than all other algorithms I chose. SGD is good regression helper, but for classification, it seems quite unstable in my experiment, the test accuracy result I get could be vary from 0.4 to 0.55 for the same parameters, it depends on linear SVM in my model. SGD classifier just use the mini-batch techniques to improve the convergence and improve the result. Also, the parameters are too many to well-tune the good performance SGD classifier, it will be hard to get a good result by it. For the confusion matrix, we could find it the common case that the class 2 is almost confused with both sides, but the convergence makes the class 1 and class3 has very good result like most of other algorithm does.

For the Random Forest, it performs better than the trivial system and the baseline system a lot. Also, we could find that the test accuracy and the macro f1 score is the highest one in my experiment. For the reason, I think random forest is based on each of the features of dataset, it will randomly define the decision trees and split the features nodes by nodes. The random choose datapoints from the sample by each of the trees could make the situation become general and will remove the problem of outlier and some overfitting problem. The sparsity and the unbalanced data could not easily influence the result, because RF will small the error. Also, random forest could easily find the feature

importance and the dependency between each other features. We could see that the training accuracy is almost equal to 1. For the confusion matrix, the classification for the class 2 is almost 50% which is good than most of the algorithms, except for LDA and KNN.

For the MLP, it also performs better than the trivial and baseline system. We could find that, MLP performs better than all the other algorithm, except for Random Forest. Because the MLP is a kind of BP-ANN, which update the weight vector in each of the iteration by the error in each of the fully connected layer. Thus, the learning will be quite slow but fully, where we get the second best result in the table above. To compare with the other traditional algorithm, neural net is a good improvement except for the time and space consumption. In the confusion matrix, we could find class 1 and class3 has a good prediction, and class 2 is confused with them, because the class 2 is the medium between the good and bad.

5. Contributions of each team member

This is personal project.

6. Summary and conclusions

- Approach

The wine quality dataset includes 3 classes. Firstly, I create the trivial system and baseline system used to do the comparison and baseline. For the main step, basically, firstly we need to do is the data pre-processing. I firstly try the different performance of the Standardization/ Normalization methods by the cross-validation to see the difference of them and their performance and doing the analysis. Then, I experiment the influence of the imbalance and balance dataset by the cross-validation. Then, I use LDA to show the difference caused by the feature space expansion by cross-validation. Next, for feature dimension reduction, I use the Random Forest to get the result of the feature importance to see that we don't need to implement the feature dimension reduction method, because all features are important.

After finishing the pre-processing part, I use the best pre-processing combination I thought from the result to train the model by model-selection by cross-validation. Then, based on the best model I get from model selection, I use the test set to get the final result by the test accuracy, macro f1 score and confusion matrix. For this part, I implement 6 different algorithms except for the baseline and trivial system, they are LDA, KNN, SVM, SGD, RF and MLP. Based on those six models, I do the comparison and analysis for them and for the baseline and trivial system.

- The most key results I get is:

Feature expansion performs well in the model where the feature number is not too many.

Standardization/Normalization: And for those algorithms based on gradient descend, standardization will improve the performance, such as MLP and SGD; for those algorithms based on distance learning, such as KNN and SVM, standardization is very important; And for the LDA and RF, the standardization is not important and will not improve the performance significantly.

Dataset balance will help improve the learning algorithm to get a better result.

Test accuracy: RF>MLP>KNN>LDA>SVM>SGD>Baseline>trivial

Macro f1 score: RF>LDA>MLP>SVM>KNN>SGD>Baseline system>Trivial system

- For the farther improvement I think I calculate the correlation of each of the features and do the feature selection based on this. Also, I think I could try the XGBoost algorithm to see the performance. And I could farther improve the MLP or using other neural network algorithms. Or, maybe I need to do the filter to the dataset to make it performs better.

The most important thing I learned I think is the importance of reading and understand the official documents for the algorithms and reading them well in order to using them well. Then I think the open mind is quite important too, because we need to come up ideas by ourselves and make it come true by ourselves. A good idea will help a lot, although we may not get good result, we could open up more ideas.

References

- [1] UCI machine learning Repository
<https://archive.ics.uci.edu/ml/datasets/Wine+Quality>
- [2] <https://stats.stackexchange.com/questions/109071/standardizing-features-when-using-lda-as-a-pre-processing-step>.
- [3] N. F. Raoof Gholami, " Chapter 27 - Support Vector Machine: Principles, Parameters, and Applications, Editor(s): Pijush Samui, Sanjiban Sekhar, Valentina E. Balas, Handbook of Neural Computation, Academic Press, 2017, Pages 515-535, ISBN 9780128113".