COMP 6721

Mini Project 2 Report

by

Basant Singh
(Student ID: 40047623)

1. Introduction :

In the project, I have used machine learning framework to experiment with different learning algorithm and different dataset. The dataset is about the classification of black and white images.
In the following sections, I have explained steps that were analyzed for the experimentation to understand the behavior of the classifiers and tweaked its hyperparameter in order to increase its performance.
The experiment starts with analyzing and preprocessing the training dataset. I have created models using 4 classifiers(Decision tree, Random Forest, Naive Bayes, and SVM) on two datasets (i.e Training and Validation dataset) to compare the performance of models based on performance indicator (accuracy, recall, precision, and f1 score).

These models are created in form of 3 experiments as listed below.

1. Models built using the default settings in scikit-learn (not tweaking the hyper-parameters) of the classifier.
2. Models built by tweaking the hyper-parameter values.
3. Models built by finding the optimal values of hyperparameter using an exhaustive search technique called grid search.

2. Data Analysis & Preprocessing (Common to both dataset)

The first step is to analyze and preprocess the datasets as mentioned below.

- Missing entries identification: I analyzed the datasets for missing entries and found that data is clean as no null value for input features was present. This step is required to build a successful model in scikit-learn as datasets with missing entries are incompatible with scikit-learn-learn estimators which assume that all values in an array are numerical and that all have and hold meaning.
- Duplicate value removal: I performed a test to look for duplicate values in the datasets because the presence of duplicate data will lead the classifier to give biased classification results.

3. Experimentation and findings on DataSets

In the preprocessing step, one duplicate data instance was identified in Dataset1, so it was removed.

Visual representation of the distribution of the class label vrs count for the training dataset

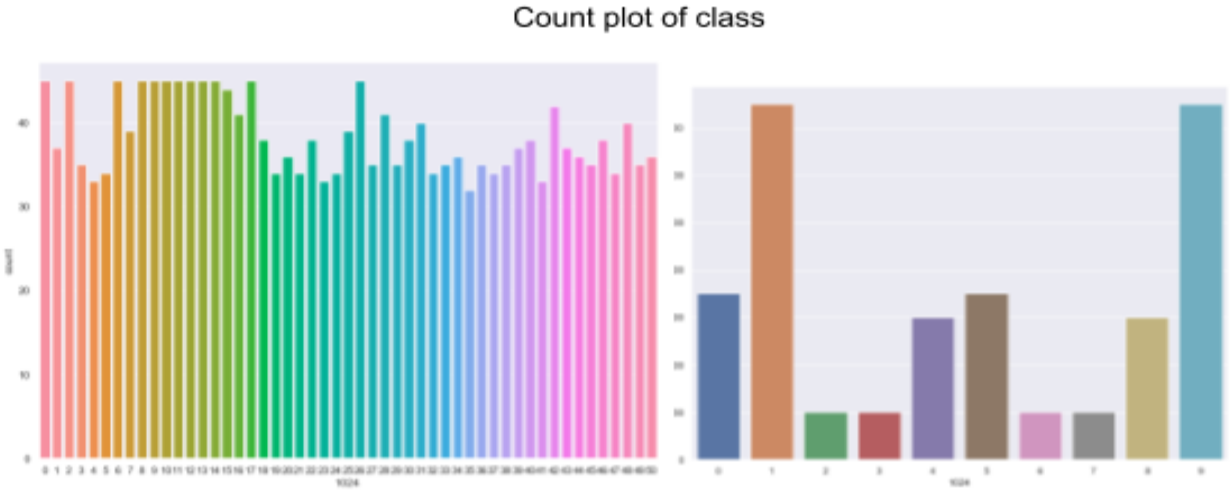

Fig1 : Dataset1

Fig2 : Dataset2

For the dataset1, In Fig 1, each class has a significant presence in the training dataset so the model will have enough class instances to learn different class labels.
For the dataset 2, In Fig 2, it can be seen that class 2, class 3, class 6 and class 7 have a relatively low presence in the dataset, so these class instances may be relatively difficult to learn.

4. Decision tree :

Experiment on Dataset1 :

Experiment 1: Model built on the default setting of hyperparameter in scikit-learn.

The model built on validation set has the accuracy of 29.3 %, average precision of 31 %, average recall of 29% and f1-score of 29%. The performance of the model, in general, was not consistent across the class. For example for class 'A' the recall was 67% whereas for class 'r' and 's' the recall and precision was 0%.

Experiment 2: Model built by changing hyperparameter values

In below table 1.1, I have recorded the change in the performance of the model based on the max_depth hyperparameters.

max_depth	accuracy(%)	precision(%)	recall(%)
10	23.7	32	24
20	28.5	31	29
30	29.1	31	29
40	29.3	31	29
50	29.3	31	29

Table 1.1

Likewise I tested for other hyper-parameters as min_sample_leaf, min_sample_split, criterion, however, the performance of the model did not change much. It may be because the effect these hyper-parameters is reflected from max_depth that restricts overfitting. So, as per table 1.1, we can observe that with the increase of max_depth from 10 to 40, performance of the model increases but further increase in depth does not result in an increase of the performance.

Experiment 3: Model built using hyperparameter tuning technique

Since there were several hyper-parameters for the classifier, so in order to tune the hyperparameters in order to find the best set of hyper-parameter values. We used a grid search optimization technique as mentioned before.

I performed grid search using below search parameters,
Grid search parameters = {'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],'max_features': ['auto', 'sqrt'],'min_samples_leaf': [1, 2, 4], 'min_samples_split': [2, 5, 10],'criterion': ['gini', 'entropy']} along with random_state = 1.

I could find the best set of hyperparameters as below:
{'criterion': 'entropy', 'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2}

Using the above-mentioned set of parameters, I created the model on the training dataset which gave the accuracy of 24.09 % with average precision, recall and f1-score as 25 %, 25% and 24% respectively on the validation set.

We can observe the performance of the model on the validation dataset that experiment 1 (where parameters are set by default) and experiment 2(where parameters which were set using grid

search) have very much similar result. The probable reason for the low accuracy of the model whose parameters were set using a grid search can be that it may have possibly overfitted the model on the training set.

So in order to reduce overfitting, I pruned the trained tree with min_samples_split = 6. Due to this the accuracy of the model went up with a marginal increment to 24.23%. The point here to note is that using pruning we could generate more general tree without compromising the performance. In other words, we can say that we generated more generalized tree without compromising the performance.

Other Experiment:
Initially, I thought not to do PCA but after seeing the low performance of decision tree model on the validation dataset I performed PCA. The intuition behind this was to remove the noisy features in the training dataset that might cause the model to perform poorly.
However, the after the experiment I found that the performance of the model did not improve. So it seems there are no noisy features that I thought of as one of the reasons for the low performance of the model.

Conclusion: We could conclude that decision tree is not able to perform well for dataset 1. The possible reason for this is the classifier itself. Decision tree easily overfits and even following pruning and PCA methods did not improve the performance of the model. So, to reduce the overfitting we could try random forest which reduces variance by training on different samples of the data.

Experiment on Dataset2 :

Experiment 1: Model built using the default setting of hyperparameter in scikit-learn.
The model was built using the default hyperparameter setting in scikit-learn and random_state =1. The built model's overall accuracy is 76.75 %, whereas average precision is 77 %, recall is 77% and f1-score is 77%. At the class level, for class 2(beta) and class 3(sigma), the recall is 43% and 53% respectively, which is relatively poor with respect to model's performance in comparison for other classes.
Experiment 2: Model built by changing hyperparameter values
I observed the dependency of the performance of the model on the hyperparameters.
Highlighted records have the highest performance measure.

max_depth	accuracy(%)	precision (%)	recall(%)
10	75	77	75
20	76	77	76
30	77.1	78	77
40	76.7	77	77
50	76.75	77	77

Table 1.2

As explained in experiment 2 for the dataset 1, we can observe in Table 1.2 that with an increase in max_depth from 10 to 30, the performance of the model increases but further increase in the depth did not improve the performance of the model.

Experiment 3: Model built using hyperparameter tuning technique

I have used search grid technique with the same search parameter used for the dataset1

Best set of parameters after grid search:
{'criterion': ‘entropy’, 'max_depth': 30, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2}
The built model with above hyperparameters shows an Accuracy of 69.5%, Precision of 71% , recall of 69 and f1-score of 70%.

The low performance of the model when built using parameters from a grid search can be due to the fact that I may not be passing a required range of values to be searched during grid search.

Conclusion:
Comparison of classifier performance on Dataset1 and Dataset 2:
In all experiments, the performance of the model trained on the dataset 2 is better than the dataset1. This may be because the size of the dataset 2 is larger than the dataset 1.

5. Random Forest Classifier

Experiment on Dataset 1

Experiment 1: Model built on the default setting of hyperparameter in scikit-learn.
The accuracy of the model on the validation set is 41.2 %, whereas average precision is 42 %, average recall is 41% and f1-score is 40 %. The performance of the model, in general, is not consistent across the class. For example, at the class level, for class ‘c’, the precision and recall is 20% and 18% respectively whereas for class ’q’ precision is 100%.
Experiment 2: Model built by changing hyperparameter values
The random forest-based model was initially built with some randomly selected values for the hyperparameter and then I re-built the model by changing hyper-parameter like criterion from ‘entropy’ to ‘gini’, max_depth and n_estimators to different range of values and checked the accuracy. The performance evaluation like the accuracy of the model increased up to 62 %.
In below table, I have reported the performance of the model on n_estimators and max_depth.

n_estimators	max_depth	accuracy(%)	precision (%)	recall(%)
200	10	57	58	57
200	20	60	63	60
200	30	59.3	63	60
100	20	59.5	63	60
400	**20**	**62.2**	**66**	**62**
600	20	60.3	65	60

Table 2.1

Table 2.1, shows that the performance of the model depends on the hyperparameter like n_estimator and max_depth. I have tested models on other parameters like min_sample_leaf, min_sample_split, criterion but did not see a major change in the performance.

The dependence of performance on n_estimators: n_estimators is a number of a decision tree which are generated on bootstrap dataset ie. random subsample of the training dataset. Bootstrapping decrease the variance without increasing the bias.
Effect of the dependence of performance on max_depth has been explained in the decision tree.

Experiment 3: Using a hyperparameter tuning technique
Using ‘grid search’ technique with below range of search parameters.

Grid search parameters = { 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None], 'max_features': ['auto', 'sqrt'], 'min_samples_leaf': [1, 2, 4], 'min_samples_split': [2, 5, 10], 'n_estimators': [50, 100, 200, 400, 600, 800] }

After hours of waiting I got optimized set of parameters after grid search as :
{'criterion’ ‘entropy’, 'max_depth': 60, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2 , n_estimator : 700, random_state: 2}

With the above value of parameters, I got the accuracy of the model as 60.5 %, average precision is 65 %, recall is 61% and the F1 score is 60%.
Now if I compare the result of the Random forest with the decision tree, the following observation can be made: a)The overall performance of the model increased b)In a decision tree, for instance, the recall for class ‘r’ and ‘s’ was 0 whereas in the random forest the model to predict class these class with an average recall of 43% and 60%.

The reason behind the better performance of random forest over decision tree can be explained based on the fact that a decision tree is created by splitting the data feature values , it closely models the training dataset and may not able to extract the generalized behavior of the dataset, in another word it may overfit the trained data.
Whereas random forest creates a number of the decision tree from the same training dataset and models the general behavior of dataset. Each decision tree is created by taking a random subsample of the original training dataset. So, it avoids overfitting and creates a generalized model.

Experiment on Dataset 2

Experiment 1: Model built on the default setting of hyperparameter in scikit-learn.
The Model built has an accuracy of 85.6% and its precision is 85, recall is 86 and f1-score is 85 %.
The precision and recall of classes vary from 76% to 90% and 55% to 95% respectively.
This high performance of the model on the dataset 2 with respect to the dataset 1, can be possible because of clean separation of the classes and second reason can be because of the high number of training instance and third due to the robust classifier.

Experiment 2 : By changing hyperparameter values
Below table shows the change in performance measure with a change in hyperparameter.

n_estimators	max_depth	accuracy(%)	precision (%)	recall(%)
200	10	82.0	84	82
200	20	88.2	88	88
200	40	88.8	86	95
200	50	88.8	89	89
100	40	87.9	88	88

300	20	88.2	88	88
400	20	88	88	88

Table 2.2

In above Table 2.2 , we can observe that the model of the classifier depends on hyper parameter n_estimators and max_depth in same way as it happened for dataset 1. The major difference in the performance measure which goes up to 88.8 %.

Experiment 3: Using a hyperparameter tuning technique

Using 'grid search' technique with below range of search parameters:

Grid search parameters = { 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None], 'max_features': ['auto', 'sqrt'], 'min_samples_leaf': [1, 2, 4], 'min_samples_split': [2, 5, 10], 'n_estimators': [50, 100, 200, 400, 600, 800] }

After hours of waiting I got optimized set of parameters after grid search as :

{'criterion' gini, 'max_depth': 60, 'max_features': sqrt, 'min_samples_leaf': 1, 'min_samples_split': 2 , n_estimator : 200, random_state: 1}

With the above value of parameters, I got the accuracy of the model as 88.8 %, average precision is 89%, recall is 89% and the F1 score is 88%.

Conclusion:

Comparison of a Decision tree with Random Forest

Random forest uses bootstrap aggregation method to generate a number of an uncorrelated decision tree, each decision tree is generated from a random subsample of the original dataset. It improves performance by reducing the variance in the decision tree classifier.

As we can see from experimentation, for example, for dataset 1, in term of accuracy, decision tree have it at most 30% result whereas Random Forest has twice better performance.

6. Naive Bayes:

Naive Bayes classifier takes two parameters, priors, and smoothing. For both dataset(dataset 1 & 2) I don't feel any need to set 'priors', as it is required for highly skewed dataset or when there is need to show the importance of a class to other. In the dataset 1, classes are well balanced in terms of counts whereas in the dataset 2, number of the class instance has variation but it is not highly skewed and in addition to it, in both datasets, all classes are equally important. However, the second parameter 'smoothing' can be varied and in below experiment result, I showed how classifier performance depends on it.

Experiment on Dataset 1

Experiment 1: Model built on the default setting of hyperparameter in scikit-learn.

The model built on the training dataset shows the accuracy of 43.7 % whereas its precision, recall, and f1-score is 50 %, 44%, and 43% respectively. It was able to perform fairly with respect to other classifiers in this type of experiment(i.e experiment 1). The performance of the model, in general, is not consistent across the class but better than other classifier's model.

Experiment 2: Model built by changing hyperparameter values

In this experiment, I built a model to check its performance for the various range of values using the smoothing parameter.

var_smoothing	accuracy(%)	precision (%)	recall(%)
.05	59.7	63	60
.06	60.3	63	61
.07	61.4	63	61
.08	61	63	61
.1	60.7	63	61

Table 3.1

In table 3.1 I can observe that the as var_smoothing increase from .05 to .1, the performance of the model increases and is maximum at .07 and henceforth it decreases. One of the interesting observation for class 'z' that the model's precision and recall was 0 at all the value of var_smoothing.

Experiment 3: Using a hyperparameter tuning technique

I checked in sklearn gridsearchcv but I did not find grid search for GaussianNB. The possible reason I can understand that the grid search technique cannot be applied for Naive Bayes as it does not have any hyperparameter that can be tweaked to increase performance. In the above experiment, I have used 'smoothing' parameter to show how performance depends on it. However, It does not contribute to performance but it affects the performance of the model and we need to find optimum value of smoothing parameter that can fix a larger problem which it is intended for i.e. to adjust data so that zeros become some more or less arbitrary small values.

Experiment on Dataset 2

Experiment 1: Model built on the default setting of hyperparameter in scikit-learn.

The model built on the training dataset using the model shows the accuracy of 64.2 % whereas its precision, recall, and f1-score is 73 %, 64%, and 64% respectively.The performance of the model, in general, is not consistent across the class but better than other classifier's model..

Experiment 2: Model built by changing hyperparameter values

In this experiment, I built a model to check its performance on a various range of values for smoothing parameter

var_smoothing	accuracy(%)	precision (%)	recall(%)
0.1	76.1	77	76
0.2	76.8	77	77
0.4	76.5	78	77
0.6	75.9	78	76

Table 3.2

In table 3.2 I can observe that the as var_smoothing increase from .1 to .6, the performance of the model increases and is maximum at .2 and henceforth decrease. One of the interesting observation for class 'z' that the model's precision and recall was 0 at all the value of var_smoothing.

I can find the smoothing value for the dataset 2 is quite higher than in the dataset 2 for the reason that the count of the classes in dataset two is not evenly distributed.

7. SVM:

Experiment of Dataset1 :

Experiment 1: Model built on the default setting of hyperparameter in scikit learn.

The model on the validation dataset shows the accuracy of 50 %, precision of 51% recall of 51% and f1-score of 47%. All performance evaluation parameter was 0 for class 'z'. And the performance of the model is not consistent across the class.

Experiment 2: Model built by changing hyperparameter values

In this experiment, I tried to build the model and checked its performance on a various range of values for C and gamma hyperparameter

C	gamma	accuracy(%)	precision (%)	recall(%)
1	.001	50	51	51
10	**.001**	**66.9**	**68**	**67**
20	.001	66.5	66	66
10	.1	10	21	11
10	.01	65.7	68	66
10	.0001	54.6	57	55

Table 4.1

The behavior of the classifier is dependent on hyperparameter C as it is a trade-off between training error and the test error. The larger C is, less the final training error will be. So if we have a small value of C the training error will be high but as I increase the value of C the training error gets reduced and we can see, a similar pattern is observed in above table 3.2. However, if I keep on increasing C, there is no major improvement in performance whereas it risks losing the generalization properties of the classifier, as it will try to fit as best as possible all the training points

When gamma is very small, the model is too constrained and cannot capture the complexity of the data. If I set gamma too large, it will end up overfitting. The optimal value of gamma depends entirely on the data, in the above table I can observe, the model performance is best at .001 and after that, its performance decreases as the model starts overfitting.

Experiment 3 : Using hyperparameter tuning technique

After that, I performed the hyperparameter optimization technique known as called ‘grid search’ with below range of search parameters.

Grid search parameters :
C = [0.001, 0.01, 0.1, 1, 10] gammas= [0.001, 0.01, 0.1, 1] param_grid = {'C': C, 'gamma': gammas}

After hours of waiting, I got an optimized set of parameters after grid search as :
The best parameters are {'C': 10, 'gamma': 0.001}

As displayed in above table 3.2, for C = 10 and gamma = .001, the accuracy is 70% and precision is 68 and recall is 67%.

Experiment of Dataset2 :

Experiment 1: Model built on the default setting of hyperparameter in scikit-learn.
The model on the validation dataset shows the accuracy of 88.55 %, precision of 88% recall of 89% and f1-score of 88%. The model performance across the several classes was good.

Experiment 2 :Model built by changing hyperparameter values
In this experiment, I built a model to check its performance on a various range of values for C and gamma parameter.

C	gamma	accuracy(%)	precision (%)	recall(%)
1	.001	88.6	89	89
10	.001	90.6	91	91
20	.001	90.4	90	90
10	.1	33	30	34
10	.01	94.15	94	94
10	.0001	90.6	91	91

Table 4
.2

In the above table 3.2, we can observe the classifier’s performance on hyperparameter C and gamma, For hyperparameter C, when C is set to 1 and gradually increased to 20(keeping gamma as .001) the performance of the model increase to max at 10 after that there are no much improvement in the performance, whereas keeping C to 10 and varying gamma from .1 to .0001, I can observe the best performance of model when gamma is .01. In fact, at this value for hyperparameter, the model’s performance result is best wrt all model I have seen till now for the dataset 2.

Experiment 3 : Model built by using a hyperparameter tuning technique
After that, I performed a hyperparameter optimization technique known as called ‘grid search’ with below range of search parameters.

Grid search parameters :
C = [0.001, 0.01, 0.1, 1, 10] gammas= [0.001, 0.01, 0.1, 1]

param_grid = {'C': C, 'gamma': gammas}
After hours of waiting, I got an optimized set of parameters after grid search as :
The best parameters are {'C': 10, 'gamma': 0.001}
As displayed in above table 3.2, for C = 10 and gamma = .001, the accuracy is 70% and precision is 68 and recall is 67%.

8. Comparison:

In the above sections, we have already discussed the performance of the classifier's model on the validation dataset. In below table, I have presented the performance of the model by comparing all 3 experiments for which it has the highest performance measure.

		DataSet 1			Dataset 2			
	Experiment No.	Accuracy	Precision	Recall	Experiment	Accuracy	Precision	Recall
Decision Tree	1	29.3	31	30	2	77.1	77	78
Random Forest	2	62.2	65	62	2	88.8	86	95
Naive Bayes	2	61.4	63	61	2	76.8	77	77
SVM	2	70	68	67	2	94.1	94	94

The model built in experiment 2 and the experiment 3 has very close performance measure whereas Experiment 1 has the low-performance measure with respect to other experiments in most of the classifiers.
So, based on my observation on 4 different types of the classifier, I found SVM as best suited classifier as its model on the Dataset1 shows accuracy of 70% and on the Dataset2 accuracy is 94%.

9. Future Work:

For all classifier, the performance of the model for some classes is relatively low, which can be further investigated.
The performance of the model on dataset1 with respect to dataset 2 is always low, possibly because of a low number of training instances. So maybe we can add more data or generate synthetic data to improve the performance of the model on the dataset1.

10. References :

- https://scikit-learn.org/stable/modules/impute.html
- https://www.nltk.org/_modules/nltk/classify
- https://en.wikipedia.org/wiki/Random_forest
- https://en.wikipedia.org/wiki/Decision_tree
- https://en.wikipedia.org/wiki/Naive_Bayes_classifier
- https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html