Jyotsna Nakte jnn2078
Jairaj Tikam jpt1342

Milestone 04: Core Algorithm Fine Tuning.

Overview:
In this phase, we test our fourth candidate algorithm, trying to predict recommendation based on reviews using Neural Networks. We even try fine-tuning our core Algorithm and further discuss a detailed analysis of it comparing with the other four algorithms.
Core Algorithm: Random Forest Classifier.
Candidate Algorithms: Gaussian Naive Bayes, Logistic Regression, Linear SVC, Neural Network

I.] Candidate Algorithm Testing: Neural Networks

In this step, we tested the fourth candidate algorithm using Machine Learning Classifier the Multi-layer Perceptron (MLP) classifier. It is class of feedforward artificial neural network. It consists of three or more layers of nodes an input layer, a hidden layer, and output layer. Every node is a neuron which has weighted inputs called as synapses that uses a non-linear activation function except for input nodes. The input nodes send the information to hidden layers, and the hidden layer sends information to the output layer. It works like a linear classifier to classify data which is not linearly separable. Here, we made use of python package **MLPClassifier** from **sklearn.neural_network**. In this classifier, the model optimizes the log-loss function using LBFGS, which is an optimizer in the family of quasi-Newton methods. We mention the hidden layer size, alpha for regularization, and random state seed for random number generation. This algorithm works better for the model as it can learn non-linear models in real-time. For this algorithm, the accuracy found was **85.610%** with a recall rate of **0.927.**

```
from sklearn.neural_network import MLPClassifier
NN = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), random_state=42)
modeltesting(NN, "NN", X_train, y_train, X_test, y_test)
```

```
def modeltesting(mod, model_name, x_train, y_train, x_test, y_test):
    mod.fit(x_train, y_train)
    predictions = cross_val_predict(mod, x_test, y_test, cv = 5)
```

II.] Core Algorithm Fine Tuning:

Steps used to fine-tuning the model: As discussed in the previous report, results showed the recall being high in the candidate algorithm because of imbalanced class. As mentioned in the prior phases, the recommended class 1 occupied 82% of the data. As the precision increases recall is less and vice-versa. The accuracy, thus, could be improved by using balanced classes. For balanced classes, we decided to perform oversampling the class 0 the minority class or under-sampling to class 1. We even cleaned the data at some extent by lemmatizing to reduce the inflected words properly ensuring that the root word belongs to the language.

- **Oversampling:** Oversampling is a way known to improve models trained on imbalanced data. It is used to balance the class distribution over the dataset. When the amount of information is inadequate, the oversampling technique attempts to adjust by augmenting the size of rare samples. To perform oversampling we made use of python package

SMOTE from **imblearn.over_sampling.** The SMOTE technique is a method that instead of merely duplicating entries, it creates entries that are interpolations of the minority class, and it even under-samples the majority class. Using smote for oversampling improves decision boundaries giving better results. This strategy oversamples the minority class by producing synthetic samples by computing the k closest nearest neighbors. The techniques avoid loss of information and even Mitigates data overfitting caused by oversampling. Therefore, the results found below after oversampling show better outcomes.

```
from imblearn.over_sampling import SMOTE
X_sm, y_sm = SMOTE.fit_sample(X, y)
X_train_sm, X_test_sm, y_train_sm, y_test_sm = train_test_split(X_sm, y_sm, test_size=0.3, random_state=100)
```

- **Under-sampling:** The technique balances the imbalance dataset by reducing the size of the dominant/majority class. Class irregularity brings biased results inclined towards the majority class, reducing the classifier performance and expanding the number of false negatives. A straightforward under-sampling method is used to reduce the abundant class arbitrarily and uniformly. This method can cause a loss of important information or data. The run-time of the algorithm can be improved by decreasing the amount of training dataset. It helps in solving memory-related problems. To perform undersampling, we made use of python package RandomUnderSampler from imblearn.under_sampling. RandomUnderSampler uses an easy path to balance the data by randomly selecting a subset of data for the targeted classes. It even allows to bootstrap the data by setting replacement to True to make the algorithm run faster. The resampling with multiple classes is performed by considering each targeted class independently. It even allows for heterogeneous sampling data. In this case, using the technique loses information and does not show better results.

```
from imblearn.under_sampling import RandomUnderSampler
randum = RandomUnderSampler(random_state=0)
X_resampled, y_resampled = randum.fit_resample(X, y)
X_train_un, X_test_un, y_train_un, y_test_un = train_test_split( X_resampled, y_resampled, test_size=0.3, random_state=100)
```

III.] Detailed Analysis of Core Algorithm:

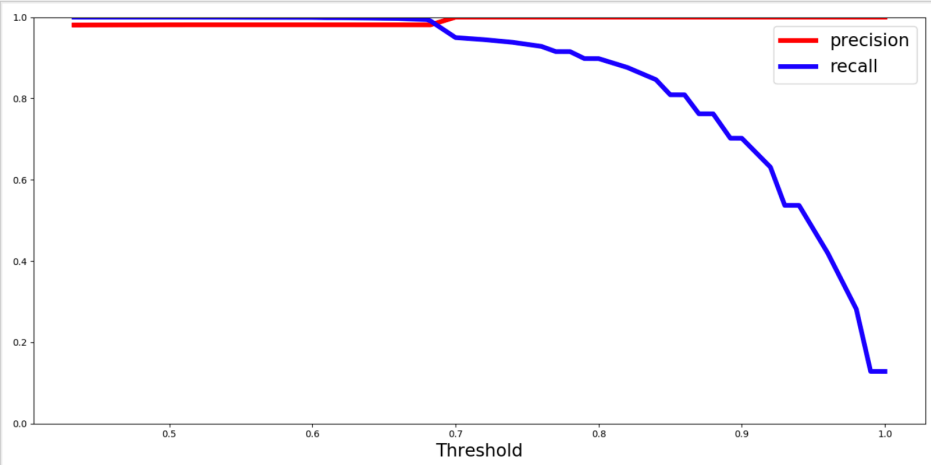
The core algorithm used here is the random forest classifier. As explained in the previous phase, this algorithm was chosen because it works efficiently, and it is scalable for large amounts of high dimensional data. Random forest classifier has no requirement of feature normalization, and it reduces over-fitting. It handles a count of thousands of input variables without variable/feature deletion, and our dataset contains 880 input features. It even works for estimating missing data and maintains accuracy when a large proportion of the data missing. Our dataset includes 3% of sparsity, and the classifier gives one of the best results. The neural network offers almost the same accuracy, but in the neural network, you have hidden layers which sometimes does work well for large dataset.

The algorithm gave an accuracy of 85.41% with precision 0.86 and recall rate 0.9792. After fine-tuning the algorithm by fixing the class imbalance, the accuracy obtained was 89.24%, which is the highest accuracy amongst all the algorithm. Here, the oversampling clearing improves the results. After testing the under-sampling technique, the accuracy reduced to 78.91%, which

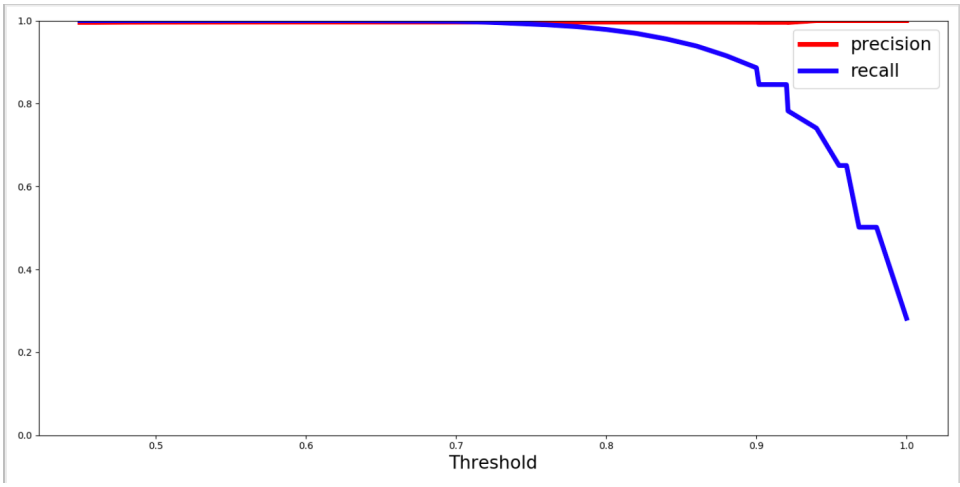
indicates the loss of important information or features in the process**.** Therefore, for fine-tuning, we use oversampling, and it improves the supervised classification model.

Further, we analyze the recall, precision rates of the dataset before and after sampling. Precision-Recall curve is the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds. The precision-recall curve is the plot of the precision on the y-axis, and recall on the x-axis for different thresholds. The plots below show that the curve is almost similar before sampling as the imbalanced dataset does not impact them. The classifier with the perfect performance level indicates overlapping of two straight lines from the top left corner to the top right corner and further down to the endpoint. Both the curves show performance nearing to perfect classifier overlapping each other with low false positive and true negative counts. The recall rate after oversampling decreased from 0.9792 to 0.899. The precision rate of the model increased from 0.86 to 0.8883. The changes in the rate and improvement in precision can be seen through the graphs.

Recall-precision curve before oversampling:



Recall-precision curve after oversampling:



Jyotsna Nakte jnn2078
Jairaj Tikam jpt1342

IV.] Detailed comparison of the results obtained:

Results after cleaning, pre-processing the dataset:

Algorithm	Accuracy	Confusion Matrix	Recall	Precision	f-score
Gaussian Naive Bayes	61.70%	706 1655 145 2192	0.5697	0.93	0.70
Linear SVC	86.07%	424 227 427 3620	0.9409	0.89	0.92
Random Forest Classifier	85.41%	264 98 587 3749	0.9792	0.86	0.91
Logistic Regression	84.92%	285 77 566 3770	0.9799	0.869	0.91
Neural Network	85.610%	455 280 396 3567	0.927	0.900	0.913

Results after oversampling the minority class of the dataset using SMOTE:

Algorithm	Accuracy	Confusion Matrix	Recall	Precision	f-score
Gaussian Naive Bayes	83.20%	4964 1145 801 4679	0.8033	0.8538	0.8278
Linear SVC	87.53%	5093 773 672 5051	0.8672	0.8825	0.8748
Random Forest Classifier	89.24%	5107 588 658 5236	0.899	0.8883	0.8936
Logistic Regression	87.634%	5080 748 685 5076	0.8715	0.8810	0.8763
Neural Network	86.176%	5059 896 706 4928	0.8461	0.8746	0.8601

Jyotsna Nakte jnn2078
Jairaj Tikam jpt1342

Results after under-sampling the majority class of the dataset using Random Sampler:

Algorithm	Accuracy	Confusion Matrix	Recall	Precision	f-score
Gaussian Naive Bayes	73.12%	1078 498 175 753	0.6019	0.811	0.6911
Linear SVC	81.98%	1018 216 235 1035	0.8273	0.8149	0.8211
Random Forest Classifier	78.91%	1008 283 245 968	0.7737	0.7980	0.7857
Logistic Regression	83.58%	1044 202 209 1049	0.8385	0.8338	0.8361
Neural Network	80.40%	981 242 272 1009	0.8065	0.78766	0.7969

V.] Analysis of the Algorithms comparing the results:

To summarize analyzing the overall results oversampling improves the models giving better accuracy. The precision and recall rates got lower after class balancing. The Naive Bayes classifier showed a significant improvement in both sampling methods making the accuracy go from 61.70% to 83.20% and 73.12%. It showed a difference of increase in efficiency between 12-22%. After balancing the classes distribution, it showed significant improvement. This behavior is due to the inherent properties of the Bayes' Theorem. Linear SVC did better without any modification with the data. The trade-off between the extra effort for oversampling of data and the 1% increase in accuracy is not worth it. Linear SVC does not rely too much on the class priors to make the predictions. In a similar tune, logistic regression also dropped to 84% accuracy with under-sampling and had only a 2% increase with oversampling. The neural network also showcased results likewise wherein the accuracy dropped by 5% to 80%, and gains were insignificant. Linear SVC, Neural network, and Logistic regression did not need resampling of data for the best optimization. The point made earlier in the report about loss or reduced information in case of under-sampling the data and accuracy being low due to this stands true. Our core algorithm – Random Forest classifier, is most affected by the reduction in information due to under-sampling. The accuracy went from 85% with original class priors to 78% when under-sampled. Our core algorithm had the highest accuracy of 89% after oversampling of data to have classes balanced.

VI.] Python Packages imported:

The below image indicates the list of all the python packages used in the complete process which was missed out in the previous phase reporting details.

```
# Dealing with data frame
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# NLP
import nltk
nltk.download('wordnet')
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

#Classification Algorithms
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import LinearSVC

# Classification evaluation metrics
from sklearn.metrics import precision_score, recall_score,accuracy_score
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix

#Data Splitting
from sklearn.model_selection import train_test_split

# Sampling of the dataset for fine tuning of the model
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
```

References:

[1.] https://imbalanced-learn.readthedocs.io/en/stable/under_sampling.html
[2.] https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html
[3.] https://www.kdnuggets.com/2017/06/7-techniques-handle-imbalanced-data.html
[4.] https://www.marcoaltini.com/blog/dealing-with-imbalanced-data-undersampling-oversampling-and-proper-cross-validation