SURABHI AMIT CHEMBRA
Dallas, TX-75056
Phone (Primary): 469-605-0961
Phone (Home): 214-454-6494
Email: surabhivasudev1993@gmail.com
LinkedIn: https://www.linkedin.com/in/surabhi-amit-chembra
GitHub: https://github.com/SurabhiAmit

Write-up

Binary Classification Task:

Introduction and code overview:

The given dataset had 40k samples for training and 10k samples for testing. There are 100 features out of which 4 features have string values and 2 features have "$" or "%" sign. The following steps were done as part of data cleaning:

- Text processing for string columns – strings were converted to lowercase, extra whitespaces were removed, misspelled or abbreviated words were clustered together like thurday or thur for thursday.
- The columns of x41 and x45 were stripped of white spaces, $ sign and % sign to convert those columns into numeric datatype.

After data cleaning, encoding of categorical values was performed. Both LabelEncoder and OneHotEncoder were tried and based on performance from logistic regression base model, LabelEncoder was chosen.

Outliers were detected and handled based on Inter Quartile Range. Any value that lies outside the minimum and maximum of the boxplot were capped to the minimum and maximum values, respectively.

KNN imputation was used to handle missing values. This imputation method considers 100 nearest neighbors with respect to other samples with similar non-missing feature values to fill a missing value.

After imputation, z-score normalization was used as the features were mostly gaussian.

The training data preprocessed as per the above method was then split to train and validation datasets in 80%-20% ratio using stratified sampling. Since the data available to train the model was heavily unbalanced (80% of data were classified as 0 and rest as 1), I used SMOTE oversampling technique on train dataset.

The following models were attempted on the training dataset with grid search and 10-fold stratified cross validation, followed by out-of-sample testing on validation dataset (test dataset did not have y-values so the classifier performance cannot be evaluated on it):

1. Neural Network (Multi-Layer Perceptron)
2. Logistic Regression
3. Support Vector Classifier
4. Random Forest Classifier
5. Ridge Regression
6. Naïve Bayes Classifier

The models are implemented using scikit-learn [2] library and though the neural network was initially developed using Keras, the long execution time (CPU based) while performing grid search on Keras model motivated me to look into the faster scikit-learn for multilayer perceptron.

Models finally chosen:

After performing selective grid search, the following models were chosen, since they exhibited commendable performance over other models attempted:

1. Model 1: Neural networks
2. Model 2: Support vector classifier

Please find the results for the test data attached as results1.csv and results2.csv using model 1 and model 2, respectively. The model trained on given data was chosen over the one trained on oversampled data after evaluating the performances.

Description and comparison of methods used:

Neural network is complex enough and the weights of interconnections between multiple layers and bias of nodes can easily digest the complexity of the data as compared to simple models like logistic regression. Therefore, it was able to train well on the given data and output accurate predictions.

SVC introduces a hyperplane boundary to divide the positive and negative instances and, in this case, radial basis kernel used will be able to find a finer boundary in a higher dimension as compared to linear SVC. This increases the rate of true positives and true negatives. I did not strive for 100% training accuracy and settled for 99%, because the validation accuracy was highest in the former case, 98.8%. Further tuning to the training set can induce **overfitting** where SVC will strive to find a hyperplane that accurately describes all training data points which will, during testing phase, lack generalization.

Comparing the above 2 methods, NN has an input layer, at least one hidden layer and an output layer. For linearly separable data, the hidden layer is not needed, and a standard perceptron can suffice. In this case, I used 1 hidden layer to optimize performance. Each feature forms a node in the input layer. Each class forms a node in the output layer. Every node in one layer is connected to every node in the next layer and the weights of each of these connections are set using **gradient descent** and **backpropagation**. The **weight changes** are computed by multiplying the gradient with the **learning rate** and summing it up with the product of the previous weight change and **momentum**.

Smaller learning rates help in taking smaller steps to avoid overshooting the error function minima, but increase the time needed to converge. In this case, relatively higher learning rates were able to find global minima without overshooting in less time. Momentum attenuates the oscillations in the gradient descent and prevents converging to local minima. Higher momentum values create instability by overshooting the minimum and very small values need not reliably avoid local minima. So, apt selection of learning rate and momentum is required for the algorithm to perform its best and grid search was used in this case to do the same.

A node gets activated when the weighted sum of its connections exceeds its **activation threshold** (bias). The number, size and activation function of hidden layers, and the values of learning rate and momentum are the hyperparameters that can be tuned. The complex structure of the neural network makes it prone to **overfitting** the training data. To prevent this, **early stopping technique** was used where a part of training data is set as validation set and when the performance of the model worsens on this validation set, the training is stopped irrespective of the number of epochs(iterations) done. The loss function used in MLP (cross-entropy) was

well-tuned by feeding in class weights to the model to take care of the unbalanced dataset, thus helping obtain better recall and lesser false negatives, given the training data is skewed towards class 0 (negative).

Another technique utilized is **regularization**, where a penalty is imposed on the weights of the nodes to avoid the model **overfitting** the training data. This gives a simpler model with a small margin of error (slightly underfits training data) and so the cost-sensitive classifier was subjected to regularization loss in this case. During grid search, it was noticed that introduction of more hidden layers overfitted and gave more false positives and false negatives on validation set. When I tried reducing the number of neurons (from 100) in the hidden layer, the model became too simple to model the complexity of data and the performance was deteriorated. I also attempted stochastic gradient descent optimizer instead of Adam, but the performance was similar with no commendable improvement/deterioration. Similarly, whether the learning rate is kept constant or set adaptive to the training process did not invite any changes in performance point of view. However, changes in learning rate parameter gave commendable improvement, and when I used a higher rate, the model was able to learn better and faster in the set number of iterations.

Regarding SVC, the model maps the training data into a space with separate categories divided by a gap(margin) with maximum possible width (**maximum-margin hyperplanes**), and validation instances are mapped to this same space and their class is predicted based on which category they fell into. SVM deals with **non-linear data** using **kernel trick** (rbf is used here) which maps the linearly non-separable features into high dimensional feature spaces. The SVMType used is C-SVC (classification) and cost is set to 1.0. Since SVC assumes that the input features are normalized and to avoid SVC getting **over focused** on dimensions with large range of values, the features of both the datasets were **normalized** prior to training.

<u>Pros and cons list</u>

- The rbf kernel of SVC is in the form of a **gaussian** function and mapping the given dataset with more than 90% of the features being normal/gaussian to a higher dimensional space by rbf kernel gives good modeling and commendable performance, as compared to MLP. So, I expect SVC (results2.csv) method to excel more.
- **SVC** is best suited when the dataset is relatively small. Quoting the **Occam's Razor**, in any algorithm, it is always best to start from the simple hypothesis and get more complex if the hypothesis cannot fit the complexity of the data. Simpler models like SVC are easier to build, validate, interpret, and improve as compared to deep learning techniques.
- MLP is capable of modeling **very complex** tasks which SVC cannot handle with its simpler structure.
- When the number of features is much larger than data samples, **curse of dimensionality** takes its toll, and performance of models like SVC which try to model the non-linear decision boundaries gets impacted. NN also gets impacted by this issue.
- NN needs huge training data to build its **complex black-box** model perfectly and hence when the dataset is relatively small, I would rather rely on SVC.
- SVCs are much **slower to train** as compared to NN. This is because, SVM training requires solving the associated Lagrangian dual (rather than primal) problem. This is a *quadratic* optimization problem in which the number of variables is very large and

equal to the number of training instances. Setting higher values for gamma increases the training time even more. The training time is high for NN as well when the number of hidden layers and neurons were increased, but lesser than of SVC.

- Regarding the **real time prediction speed**, SVC is faster than NN. This is because once the trained model is ready, SVC classification involves only the determination of on which side of the decision boundary a given point lies (cosine product). Whereas, NN prediction involves successive multiplication of the data vector with the weight matrices of the interconnections between the layers and so on, which takes substantially higher time than SVC.
- SVMs are **intrinsically two-class** classifiers, as opposed to random forests or neural networks, and hence perform better in this case of binary classification. However, for multi-class classification, plain SVC might invite performance issues (but, currently there are adaptations of SVC technique available optimized for multi-class tasks).
- There is **more control to SVC** technique via **tuning**. The kernel used, its parameters and the parameters of the maximum margin hyperplane can be tuned well which would substantially alter the SVC model built. Methods like logistic regression offers little to no control over the way model is built (very simple models). Though NN also provides control over how many layers to build and cost function, etc. the impact of tuned parameters on model is very high for SVC (presumably because the kernel selection alters the fundamental mapping of the model itself).This can explain why SVC performs well when well-tuned using grid search.
- SVM gives **support vectors**, which are the points in each of the classes most close to the boundary. These values help a lot in interpreting the model. Random forest gives feature importance which also helps understand the model better. But, NN is more complex and less sharing about its internal structure in a comprehensible manner to **business partners**.
- Methods like neural network or random forest **gives the probability** of belonging to a class. SVC gives the distance to the boundary and if probability is required, we must convert the relative distances somehow into probability. For scikit-learn models, predict_proba() function does this for us.
- Neural networks are **highly scalable**. Since SVC intermediate steps involves highly complex calculations involving construction of matrices of the size of data points, SVC requires large memory spaces and high computational complexity. So as a rule of thumb, SVC is not scalable and for real life big data problems, SVC might cause issues unless the large number of features used are homogenous with meaningful distances [like image pixels].

Notes:

- I used predict_proba() function of scikit-learn to determine the probabilities in results.csv files. However, please note that as per the user guide of scikit-learn, there can be some validation issues for SVC classifier probabilities [3]. Though SVC also provides decision_function(), it outputs signed values which only signals the likelihood of classes and not the normalized probabilities.
- I had planned to plot model complexity curve to demonstrate the overfitting and underfitting regions, however, due to long execution time of grid-search on my low-

memory personal machine, I was not able to. Please reach out to me for further analysis on bias-variance problems noticed and corrected in each of the models.

The details and explanations of how the grid search and model tuning was performed on all 6 machine learning techniques are added as part of "**Model selection Process and Procedure in detail**".

Oversampling:

SMOTE oversampling was used here, which first selects a minority class instance a at random and finds its k nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b.[1]

In this use case, evaluation data says that when oversampling is performed, more samples with positive target variable get generated and the trained model hence predicts more cases as positive. This increases the count of both true positives and false positives. So out of all positive cases, more will be identified as positive, hence increasing the recall [true_positive / (true_positive + false_negative)]- from 62% to 84% for logistic regression model. But the number of negative cases identified as positive also increases and hence false positives are more in number. So, precision [true_positive / (true_positive + false_positive)] decreases. This is because among those identified as positive, more proportion is false positives than in earlier scenario. Hence, in a nutshell, oversampling improved recall but negatively impacted precision.

Evaluation metric:

To decide whether to focus more on precision or recall, we need better knowledge of the use case. For example, if it is comparatively okay to predict positive cases as negative rather than vice versa, then the model trained on provided data will suffice instead of oversampling data. This scenario is suitable for use cases where false positives incur heavy costs, for example, detecting small-scale fraud like customers claiming insurance for small damages where the claim amount is only in thousands or less. Identifying a genuine customer as a fraud will result in loosing that customer. Whereas, for use cases where when the claim amount is very huge, like millions, it is important to employ a model which is more likely to identify negatives accurately rather than detecting positives correctly, so as to save the insurance company from the loss/big-budget fraud. So, in this case, when the model outputs positive, we can be confident that the customer genuinely deserves the claimed amount. If model outputs negative, further investigation can be held prior to denial of claim as needed.

Since the use case was not clearly provided, I could not decide whether to focus on reducing false positives or reducing false negatives. So, I tuned models based on F1 score and ROC_AUC.

Model selection Process and Procedure in detail:

Models trained on given training data and those trained on oversampled data were evaluated using the metrics like AUC score, F1 score, Precision-Recall curve etc. and evaluation was done on the validation set.

1. Logistic regression: - selected because of simplicity and interpretability

Logistic regression gave precision of 79%, recall of 62%, F1 score of 69%, AUC of 91% and Precision-Recall curve value of 80%. With regards to the oversampled data, the precision obtained is 57%, recall is 84%, F1 score is 68%, AUC is 91% and Precision-Recall curve value is 0.78%. Since AUC is same, based on F1 and PR- curve value, model trained on normal data was chosen instead of that trained on oversampled data.

2. Ridge classification: - like logistic regression

This classifier first converts the target values of the binary classification task into {-1,1} and then perform regression on it. I tried this machine learning technique because in my personal experience, I have seen Ridge classifier giving better performance for binary classification task as compared to logistic regression. However, for this use case, though it gave better performance with respect to precision, overall F1 score and AUC value were higher for logistic regression.

3. Neural Network – deep learning and complex enough

Since deep learning has multiple layers and a structure complex enough to learn the intricacies of the data by manipulating the bias of nodes and weights of interconnections via backpropagation, I attempted multilayer perceptron from sklearn library for this task. As expected, it was able to handle the complexity of data and model the unbalanced dataset well enough so that false negatives and false positives are less. Grid Search was performed on the number of hidden layers and their sizes, learning rates, type of solver used, activation technique used, and on whether to keep learning rate constant or adaptive. As expected, neural network gave good precision (98%) as well as recall(95%) and an AUC of 99.13%.

4. Support Vector Classifier (hyperplane based – rbf kernel)

SVC is another classification technique that utilizes hyperplanes to find the boundary for binary classification tasks. Using the radial basis function (gaussian kernel), I expected SVC to better comprehend the exact boundary of classification for this unbalanced dataset with gaussian features and it did perform very well, giving an AUC of 99.23% on validation set.

5. Tree-based methods: - interpretable, some features are relatively more important.

Grid search was performed on Random forest classifier with 10-fold stratified cross-validation. The grid search optimized for roc auc score gave a model with the below parameters:

a. n_estimators =100 (number of trees in random forest)
b. max_features='auto', (number of features to consider)
c. max_depth=None (max number of levels in each tree)
d. bootstrap=False (sample data points without replacement)
e. min_samples_leaf=5 (minimum count of samples in any leaf)

Random trees are, generally, immune to overfitting when more than one estimator tree is used because of bagging feature and random feature selection. So, I used 100 estimators. When bootstrap is set to False, sample data points were taken without replacement and this helped to get comparatively better representation of both the classes in the unbalanced dataset.

The resultant tree-based model gave AUC score of 98% on validation set, but NN and SVC outperformed random forest in this task. This is mainly because the data is heavily skewed

towards class 0 and the random forest could not give high recall because of many false negatives.

6. Gaussian Naïve Bayes – probabilistic method (statistical)

Naïve Bayes model was also attempted; however, the performance was not good enough and the precision on validation set was only 60% and AUC score was 87%, which are comparatively less than other methods.

Learning curve-

Regarding the training dataset size, a typical rule of thumb in machine learning can be that there should be at least 5 training samples per dimension/feature. **Peaking phenomenon** or Hughes phenomenon says that when the number of training samples is fixed, the generalization power of the model first increases with dimensionality but then decreases. Learning curves helps understand if the training data size used is sufficient. And in this case, the training data was found enough sized, but the unbalanced classes were explicitly taken care of while modeling.

REFERENCES:

1. https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
2. https://scikit-learn.org/stable/
3. https://scikit-learn.org/stable/modules/svm.html#scores-and-probabilities