Credit Approval Prediction

by Sai Srinivas Murthy Narayanam

Abstract: We all have used a credit card for shopping, paying bills or for building a good credit score. It has been an essential part of our modern-day lives. In this report, we aim to predict if an applicant's credit score is good or bad based on the applicant's details such as age, work experience, number of family members, housing type etc. The dataset used to train and test the model is extracted from an open source – Kaggle. We found that the model trained can predict if the applicant's credit is good or bad with an accuracy over 97%.

1. Introduction

The aim for this project is to predict the good/bad credit standing of an applicant based on the information such as income, age, work experience etc.

The dataset consists of a wide range of applicant's data. It includes ID, gender, car, realty, children, income, income type, education, family status, housing type, age, experience mobile, work phone, telephone, email, job title, number of family members, month balance and due standing. Using all the features of the applicant's data, we would predict if the person had a good credit standing (good/bad).

The algorithm used for the prediction were SVM and KNN. Lagrangian formulation ids taken from [1] equation (9) which explains how to maximize the cost function with inequality constraints by forming an optimization problem and the stochastic gradient descent to update the weights has been referred from [2]. [2] also has advanced SGD algorithms incase the classical SGD doesn't perform well on the dataset.

I have used these specific algorithms as they both have slightly different approach to predict – SVM using distance of the datapoints to the hyperplane and maximizing it for best results while KNN is using nearest neighbors (clustering) approach to predict the class.

stochastic gradient descent algorithm is used to update the weights over 50 epochs as a part of training and the resultant weight from the algorithm is used to predict the test set outputs.

In section 2, we would be discussing in detail about the data preprocessing, data analysis followed by the split of train and test sets for the algorithm. We would also discuss more about the algorithm in detail.

In section 3, we would be discussing about the results of the training and testing of the algorithms that we used to predict good/bad credit score along with some metrics such as accuracy and F1 scores of each algorithm.

2. Experimental set-up

The dataset is derived from an open source – Kaggle. The data set consists of 2 csv files – application_record, credit_record. Both the csv files together consist of all the features mentioned in section 1. The link for the dataset is mentioned below.

https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction

Data extraction: the csv data has been read and stored in a data frame (2 csv files data into 2 separate data frames). The data frames have been given column names for ease of identification and processing the data further.

Data Preprocessing: the data in columns Age and experience are in days. So, the data in these columns have been converted to year for better interpretation of data. the data in columns age, experience and income have been normalized as an effort to bring down the number within the range of 0 to 1 so that the cost function won't blow up.

The features which are in text format have been converted to number data using encoder off the shelf function in sklearn library – Label Encoder. A column with 1s have been appended to the feature vector data frame (X) to convert the linear classifier wx + b → wx (updated/compressed version).

The column status in the data frame credit_data has 7 categories – 0, 1, 2, 3, 4, 5, C and X where 0, C, X tells us that the applicant doesn't have any overdues and 1,2,3,4,5 tells us that the applicant has due over multiple days/months depending on the category. So, we classify the data to 2 categories – good (applicant having more instances of no dues than due instances) having labelled '1' or bad (applicant having more due instances than no due instances) having labelled '-1' credit status standing and form a bi-categorical problem.

An additional column is added with details of the ratio of no due instances to due instances so that we don't lose valuable information of the number of instances in each category. After processing the data with all the above-mentioned steps, a new data frame has been formulated by merging the data in both data frames (application and credit data frames) by mapping with the applicants ID. And then indexed the data instead of identifying the data with an ID (as it has no valuable information of applicant data).

As a final step of data preprocessing, I have identified the instances which have empty columns (Nan) and removed those instances as they are not providing full information of the applicant. In machine learning terms, identifying the missing values, and deleting the rows which have missing values.

Splitting the dataset: the final dataset after all the preprocessing steps, the data is then split in 80/20 ratio as training and testing datasets using the function train_test_split function imported from the library sklearn.

Algorithm: I have implemented the SVM algorithm with stochastic gradient descent to predict the good/bad credit standing of an applicant. The concept behind SVM is to find a hyperplane/s to maximize the separation between different categories and center the hyperplane to have best results. This can be achieved by formulating an optimization problem with an inequality constraint as below –

optimizing $\mathbf{(½)}||\boldsymbol{W}||^2$ with an inequality constraint $\mathbf{Y\,(\boldsymbol{W}X + \boldsymbol{W}_0) \geq 1}$ [1]

the above problem can be solved by forming a lagrangian as cost function and implementing in python using 3 functions – lagrangian, calculate_cost_gradient and sgd.

Lagrangian[1] – implementing the lagrangian with the above-mentioned equations with a regularization parameter (lambda or C)

calculate_cost_gradient – this function is helpful for calculating cost gradient for every row (sample in the dataset) sequentially.

Sgd [2]– implementing the gradient descent using the above functions to update the weights with learning rate as the varying parameter.

The flowchart in Fig1 gives an overview of the steps implemented for SVM with SGD.

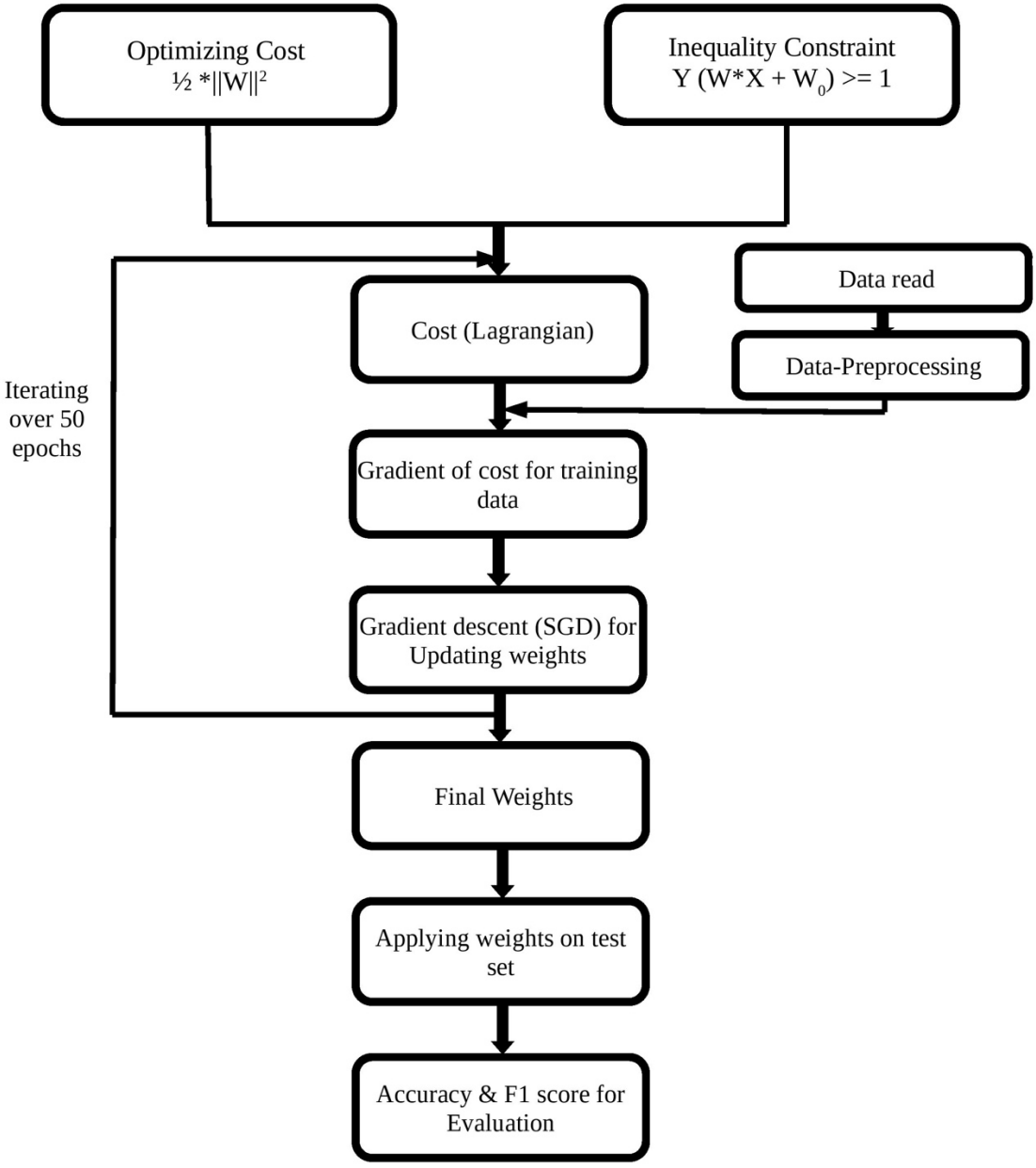

Fig1. Flow chart for SVM with SGD

As for KNN, an off the shelf function with neighbors as hyperparameter, from the sklearn library has been used and compared against the SVM algorithm implemented.

3. Results and discussion

SVM: Hyperparameters for the SVM: regularization, learning rate. I have varied both the parameter to some extent and the following are the results after hyperparameter tuning.

regularization	Learning rate	accuracy	F1 Score
1000	0.001	0.921	0.93
1000	0.01	0.554	0.7
100	0.001	0.9	0.91
1000	0.0001	0.979	0.981
10000	0.001	0.9	0.91

From the above table we can clearly observe that the learning rate = 0.0001 and regularization parameter = 1000 gives us the best results with almost 98% accuracy.

The following is the classification report with recall, precession, F1 and accuracy scores for better visualization of the model performance with learning rate = 0.0001 and regularization parameter = 1000

	precision	recall	f1-score	support
-1	0.98	0.98	0.98	2289
1	0.98	0.98	0.98	2738
accuracy			0.98	5027
macro avg	0.98	0.98	0.98	5027
weighted avg	0.98	0.98	0.98	5027

Having a high precision, recall and F1 scores means that the model is performing well on the test set. The models have been trained and test multiple times for 50 epochs and by jumbling up the training and test sets with random value initializations for the weights and observed to have almost similar performances at almost all the instances.

KNN: the sklearn function is having only one hyperparameter – neighbors. So, I have evaluated the systems by varying the hyper parameters and performed the hyperparameter tuning and got the results below.

Neighbors	accuracy	F1 Score
10	0.922	0.924
100	0.90	0.899
1000	0.847	0.837
2500	0.81	0.79

From the above table, we can observe that having the hyperparameter set to higher value gives us poor results. So, the hyperparameter Neighbor = 10 gives us the best model for the dataset.

The following is the classification report with recall, precession, F1 and accuracy scores for better visualization of the model performance with Neighbor = 10

	precision	recall	f1-score	support
-1	0.82	1.00	0.90	2289
1	1.00	0.82	0.90	2738
accuracy			0.90	5027
macro avg	0.91	0.91	0.90	5027
weighted avg	0.92	0.90	0.90	5027

Comparing both the models, we could see that the precision and recall for both classes in SVM is the same while its varying in KNN, SVM having higher precision and lower recall for bad credit status class and KNN having higher precision and lower recall for good credit status class. And most importantly, SVM has better accuracy and F1 scores when compared to KNN.

SVM outperforms the KNN for the data set after hyperparameter tuning. The next section discusses about the inferences and possible reasons for the performance differences.

Note: the models performance would vary for every training instance but the above data gives us a good understanding of a generic performance of the models.

4. Conclusion

We could draw an inference that the SVM after hyperparameter tuning performs better when compared to the KNN with hyperparameter tuning. This might be due to the way the data has been distributed / spread spatially. Since SVM optimizes the distance metric as it's a part of the cost function, it is giving us better results with for a given set of learning rate and regularization parameters.

We conclude by inferring that SVM outperforms KNN when the data is more spatially spread out and KNN performs better or equally good to SVM when the data is more clustered or densely located for multiple classes.

We could also improve the SVM performance by adopting better optimization algorithms in place of SGD as discussed in the reference paper [2] in case of any dataset preforming bad with a simple SGD SVM model.

5. References

[1] Orchel, M., Suykens, J.A.K. (2022). Improved Update Rule and Sampling of Stochastic Gradient Descent with Extreme Early Stopping for Support Vector Machines. In:, *et al.* Machine Learning, Optimization, and Data Science. LOD 2021. Lecture Notes in Computer Science (), vol 13164. Springer, Cham. https://doi.org/10.1007/978-3-030-95470-3_11

[2] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *ArXiv, abs/1609.04747*.