

# End Semester Project - Report

Kevin Sona DA24B007

November 2025

## 1 Model summary

- I choose to implement a stacked model, with 3 base models and a meta model. First, the train set was divided into K-folds. For each base model, the predictions of the  $K^{th}$  fold were obtained from a model trained on  $(K - 1)^{th}$  folds. We stack all the predictions for the K folds. We then train the meta model on these stacked predictions (out of fold predictions for the base model).

For the base models, we choose logistic regression, SVM, and XGBoost. The rationale behind choosing these models is that they cover all types of classifiers i.e, probabilistic, margin based, and tree based. The meta model is trained to choose between one of these model predictions, based on the nature of the record.

- Stacked model containing 3 base models:

### 1. Multi-class logistic regression:

- Learning rate: 0.06
- Number of iterations: 40
- Design choice: Implemented using softmax for multi-class regression

### 2. SVM

- Learning rate: 0.003
- Number of iterations: 20
- regularization term: 0.01
- Design choice: Implemented using one vs all

### 3. XGBoost

- Estimators: 12
- Max depth: 6
- Sampling rate: 0.05
- Learning rate: 0.375

- Regularization term: 1
- Gamma: 0
- Design choice: Instead of building an xgboost model for each class (as in one vs all), I built one model with one tree for each class in every estimator.
- Meta model : XGBoost
  - Estimators: 5
  - Max depth: 4
  - Sampling rate: 1
  - Learning rate: 0.375
  - Regularization term: 1
  - Gamma: 0
  - Design choice: This model doesn't need to be as complex as the base model, since it is already provided with the predictions from the base models.
- Number of folds: 2

## 2 Hyper-parameter tuning

- Since the stacked model has too many parameters that determine the run time of the model, we settle on one of the combinations that run within 5 minutes and gives maximum f1 score.

Parameters that determine run time:

1. Logistic regression: None (Since LR is not computationally expensive)
2. SVM: Number of iterations
3. XGBoost: Estimators, Max depth
4. Stacked architecture: Number of folds (K).

The rest of the parameters were set to their base values. A few combinations are highlighted below:

Iters(SVM)	Esti(XGB base)	depth(XGB base)	Esti(XGB meta)	depth(XGB meta)	Folds	F1 score
30	10	5	3	3	4	0.929
30	12	6	3	3	3	0.923
50	12	6	3	3	2	0.931
20	8	6	3	3	3	0.927
30	10	5	4	4	2	0.924
20	12	6	4	3	2	0.932

Table 1: Hyper-parameter tuning

- Next we move on to tune the rest of parameters one model at a time. First, we tune the linear regression model for learning rate and number of iterations using grid search

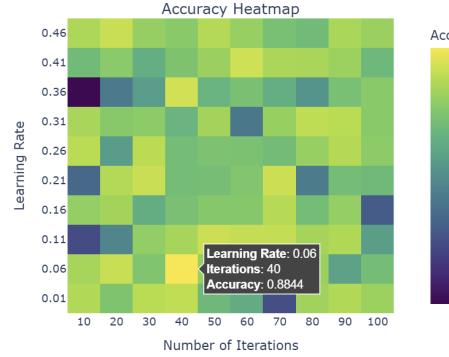


Figure 1: Heatmap for LR

Hence we settle on iterations = 40 and learning rate = 0.06

- For SVM, we tune for learning rate and regularization parameter in a similar fashion using grid search as follows:

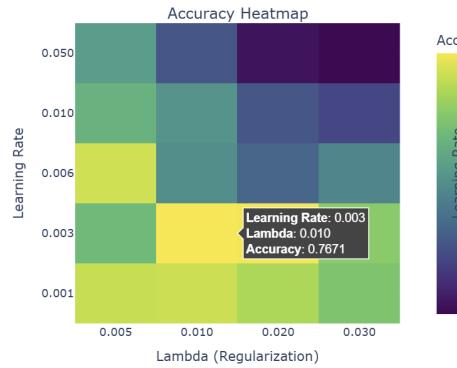


Figure 2: Heatmap for SVM

Hence we settle on learning rate = 0.003 and lambda = 0.01. We also noticed that the score started to plateau for iterations over 20.

- For XGBoost, we extend our results for tuning from the previous assignment.

### 3 Conclusion

Final model stats:

Model	F1-score
Stacked	0.932
XGBoost	0.911
SVM	0.89
LR	0.901

Table 2: Model comparison

We see that the increase in accuracy obtained by implementing the stacked model over the individual models is minimal, about 0.02. This behaviour is because among the base models, the XGboost model dominates over other models in almost all cases.

I also implemented KNN with PCA, and obtained an accuracy of 0.95. But since KNNs were not implemented in the lab, I have omitted it from my submission. I have added a KNN class to the algorithms file for your reference. In case you are running it make sure to transform all data to 30 components using PCA before training or testing it.

All the base models were trained on scaled data, which gave a small boost in the f1-score. However PCA transformations were neglected since they were observed to not give better performance.

In order to implement the stacked model, I had to rewrite a few scikit learn functions such as pipeline and stacker using numpy and pandas.