ML LAB ENDSEM

Sri Harsha V

November 15, 2025

ABOUT ALGORITHMS

PRINCIPAL COMPONENT ANALYSIS (PCA)

PCA is used for projecting high dimensional data into a lowr-dimensional subspace while preserving the maximum variance.

Given a dataset $X \in R^{n \times d}$, PCA proceeds as follows:

$$X_c = X - \mu, \qquad \mu = \frac{1}{n} \sum_{i=1}^{n} X_i$$

where $X_1, X_2, ..., X_i, ..., X_n$ are datapoints .

Mean-centering is an essential pre-processing step in PCA. It ensures that each feature has a mean of zero.If the data is not mean-centered, the first principal component may point toward the direction of the mean rather than the direction of maximum variance if the mean is high.

Then find the eigenvalues and eigenvectors of the covariance matrix using:

$$(\lambda, v) = \texttt{np.linalg.eigh}(\Sigma)$$

where $\sum$ is co-variance matrix . The eigenvectors are sorted in decreasing order of their eigenvalues, and the top k eigenvectors form the projection matrix:

$$W = [v_1 \ v_2 \ \dots \ v_k]$$

The data is then projected onto the principal components:

$$Y = X_c W$$

Here, Y is the $n \times k$ reduced representation of the data.

K-Nearest Neighbors (KNN)

KNN algorithm stores the entire training dataset. When a new sample is given for prediction, the algorithm computes the distance between the sample and all training points. In this implementation, Euclidean distance is used as the distance metric.

The k nearest training samples are then taken, and their class labels are collected. The final predicted class is obtained through majority voting among these k neighbors.

KNN is easy to implement and works well for small datasets, but it can be computationally expensive for large datasets because it requires distance calculation with all training samples during prediction.

XGBoost

XGBoost is an ensemble of decision trees sequentially, where each tree attempts to correct the errors of the previous ones. The algorithm minimizes a differentiable loss function using gradient descent while adding regularization terms to control overfitting and improve generalization.

Column Subsampling

Column sub-sampling takes some random features at every split instead of considering all available features while constructing each tree .

SVM

TUNING OF HYPER-PARAMETERS

For KNN ,

Number of Neighbors (k)	**F1 Score**	**Accuracy**
3	0.9452	0.9456
5	0.9374	0.9376
7	0.9454	0.9456
9	0.9450	0.9452
11	0.9404	0.9408

Table 1: KNN Performance for Different Values of k

By changing the k value , there is no much difference in F1 score and accuracy . So I am taking 7 as k-value since smaller k values make the model sensitive to noise, while larger values smooth the decision boundaries.

For SVM ,
$lambda_p$ is a regularisation parameter .

n_iters	**Accuracy Score**	**F1 Score**
100	0.8391	0.8357
200	0.8595	0.8602
300	0.8491	0.8512

Constant Hyperparameters : learning_rate $= 0.01$, $\quad lambda_p = 0.01$

we can observe that by changing no.of iterations there is a increase in F1 score and accuracy but then there is decrease in F1 score and accuracy .

With PCA with 500 components applied, and keeping the hyperparameters

$$\text{learning\_rate} = 0.05, \qquad lambda_p = 0.01,$$

the model performance for different values of n_iters is:

n_{iters}	Accuracy Score	F1 Score
100	0.7018	0.6944
200	0.7022	0.6952
300	0.7034	0.6970

Therefore PCA with 500 components is bad option for SVM , since it margin-dependent classifier .

With PCA with 600 components, and fixed learning_rate $= 0.05$, $\quad lambda_p = 0.01$

n_{iters}	**Accuracy Score**	**F1 Score**
100	0.703	0.695
300	0.706	0.700

Then i tried with PCA with 600 components, this one also giving poor results this is because PCA transforms the feature space to maximize variance .During this transformation, margins between classes may be distorted.Some directions important for class separation (but with small variance) can be dropped by PCA.

With fixed $lambda_p = 0.01$ and $n_{\text{iters}} = 200$

learning_rate	**Accuracy Score**	**F1 Score**
0.01	0.860	0.860
0.05	0.806	0.797
0.10	0.814	0.796
0.30	0.825	0.822

since SVM update their decision boundary based only on misclassified or margin-violating samples, which exert the most influence on the model's direction of optimization.When the learning rate is set too high, these updates can become too aggressive, causing the optimization process to overshoot the optimal margin.

With fixed learning_rate $= 0.01$ and $n_{\text{iters}} = 200$

$lambda_p$	Accuracy Score	F1 Score
0.01	0.860	0.860
0.05	0.792	0.769
0.10	0.846	0.848

from this I have taken parameters to be $lambda_p = 0.01$, $n_{\text{iters}} = 200$ and learning rate = 0.01 .

For XGBOOST with column subsampling ,

- **n_classes:** Number of distinct target classes for multiclass classification.
- **n_estimators:** Total number of boosting iterations (trees).
- **learning_rate:** Step size controlling how much each tree contributes to the final model.
- **learning_rate_decay:** Optional exponential decay factor that gradually reduces the learning rate over iterations to improve stability.
- **max_depth:** Maximum depth of each decision tree (stump). .
- ***lambda*** (λ) **(L2 regularization):** Penalizes large leaf weights to prevent overfitting and improve generalization.
- **feature_subsample_ratio:** Fraction of features randomly selected for each tree (column subsampling).
- ***gamma***(γ)**:** Minimum loss reduction required to make a split.

Fixed parameters: learning_rate $= 0.3,$ max_depth $= 4,$ $\lambda = 1.0,$ feature_subsample_ratio $= 0.04,$ $\gamma = 0.1$

n_estimators	Accuracy Score	F1 Score	Time
30	0.933	0.933	96 s
40	0.937	0.938	153 s
50	0.942	0.942	220 s
60	0.937	0.939	263 s

Increasing the number of estimators generally improves accuracy and F1 score, but after a point the gains become small. However, the training time increases steadily as more trees are added.

Fixed parameters: $n_{\text{estimators}} = 50,$ max_depth $= 4,$ $\lambda = 1.0,$ feature_subsample_ratio $= 0.04,$ $\gamma = 0.1$

learning_rate	Accuracy Score	F1 Score
0.1	0.926	0.925
0.2	0.944	0.944
0.3	0.950	0.949

As the learning rate increases, the model initially improves because each tree makes stronger updates. However, very high learning rates become unstable and may overshoot the optimal solution, causing a slight drop in accuracy. Thus, a moderate learning rate (0.2–0.3) gives the best performance.

By doing PCA on dataset,

Fixed parameters: learning_rate $= 0.3,$ max_depth $= 4,$ $\lambda = 1.0,$ feature_subsample_ratio $= 0.04,$ $\gamma = 0.1,$

PCA components=500

n_estimators	**Accuracy Score**	**F1 Score**
30	0.875	0.873
40	0.874	0.873

Fixed parameters: learning_rate $= 0.3,$ max_depth $= 4,$ $\lambda = 1.0,$ feature_subsample_ratio $= 0.04,$ $\gamma = 0.1$

PCA components=600

n_estimators	**Accuracy Score**	**F1 Score**
30	0.861	0.859
40	0.876	0.875

PCA can hurt performance because it keeps the features with the most variance, not the ones most useful for separating classes. This means important details for classification may get removed, so the model performs worse.

Fixed parameters: $n_{\text{estimators}} = 50,$ $\lambda = 1.0,$ feature_subsample_ratio $= 0.04,$ $\gamma = 0.1$

learning_rate	**max_depth**	**Accuracy Score**	**F1 Score**	**Time**
0.3	3	0.937	0.936	184 s
0.3	4	0.941	0.940	227 s
0.3	5	0.944	0.943	268 s

Deeper trees can learn more complex patterns, which often improves accuracy and F1 score. However, greater depth increases training time and may lead to overfitting due to more complex splits.

Fixed parameters: $n_{\text{estimators}} = 50,$ learning_rate $= 0.3,$ max_depth $= 3,$ $\lambda = 1.0,$ $\gamma = 0.1$

feature_subsample_ratio	**Accuracy Score**	**F1 Score**	**Time**
0.04	0.941	0.940	224 s
0.04	0.941	0.940	360 s
0.30	0.941	0.940	660 s

Using a larger feature_subsample_ratio increases the number of features evaluated at each split, so training becomes much slower. However, the accuracy remains the same, meaning the extra computation does not improve the model's performance.