Classification from Clustering Result

Case study using “Salary Prediction Classification” data

Include:

- Model comparison:
  - K-Nearest Neighbors (KNN)
  - Logistic Regression (LR)
  - Decision Tree (DT)
  - Random Forest (RF)
  - Support Vector Machine (SVM)
  - Naive Bayes (NB)
- Model evaluation
  - Confusion matrix
  - Accuracy, Precision, Recall, F1-Score
  - MSE train, MSE test, Learning Curve to detect overfitting
- Hyperparameter tuning: GridSearchCV

Data Source Overview

Source: https://www.kaggle.com/datasets/ayessa/salary-prediction-classification

Clustering result: https://raw.githubusercontent.com/nairkivm/clustering-people/refs/heads/main/clustering_result.csv

```
(28492, 9)
```

```python
salary_df.head()
```

✓ 0.0s Python

	education_num_scaled	hours_per_week_scaled	log_age_scaled	marital_status_ Married-civ-spouse	relationship_ Husband	education	age_level	hours_per_week_level	cluster
0	1.091374	-0.046192	0.155809	0	0	Bachelors	Dewasa Tengah	Waktu Penuh	0
1	1.091374	-1.915557	0.853721	1	1	Bachelors	Dewasa Akhir	Waktu Paruh	1
2	-0.412692	-0.046192	0.083079	0	0	HS-grad	Dewasa Tengah	Waktu Penuh	0
3	-1.164724	-0.046192	1.017920	1	1	11th	Dewasa Akhir	Waktu Penuh	1
4	1.091374	-0.046192	-0.768005	1	0	Bachelors	Dewasa Muda	Waktu Penuh	0

Data Preparation: Separate Features vs Target, Scale with MinMaxScaler

```
# Separate features (X) and target (y)
X = salary_df.drop(columns=['education', 'age_level', 'hours_per_week_level', 'cluster'])
y = salary_df['cluster']
```

✓ 0.0s

Scale the features with MinMaxScaler to enhance the performance model.

```
# Scale the data using MinMaxScaler
scaler = MinMaxScaler()

numeric_columns = X.select_dtypes(include=['int64', 'float64']).columns
X[numeric_columns] = scaler.fit_transform(X[numeric_columns])
```

Data Splitting

```
# Split the data into training and test sets
def split_data(X: pd.DataFrame, y: pd.Series) -> pd.DataFrame:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)
    print(f"Training set shape: X_train={X_train.shape}, y_train={y_train.shape}")
    print(f"Test set shape: X_test={X_test.shape}, y_test={y_test.shape}")
    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = split_data(X, y)
```

✓ 0.0s

```
Training set shape: X_train=(22793, 5), y_train=(22793,)
Test set shape: X_test=(5699, 5), y_test=(5699,)
```

Building the Model

We aim to use **Logistic Regression (LR)** model as this is a simple binary classification. But, we tempted to compare with other models because the dataset is relatively small.

K-Nearest Neighbors (KNN)

- **Description:** An algorithm that classifies data based on proximity to other data points in the feature space.
- **Advantages:** Easy to understand and implement, does not require assumptions about data distribution.
- **Disadvantages:** Slow for large datasets, sensitive to feature scaling.
- **Suitable Cases:** Pattern recognition, anomaly detection.
- **Unsuitable Cases:** Large datasets, data with many features.

Logistic Regression (LR)

- **Description:** An algorithm that uses a logistic function to model the probability of an event occurring.
- **Advantages:** Easy to interpret, fast for large datasets.
- **Disadvantages:** Does not work well with non-linear data, requires independence assumptions among features.
- **Suitable Cases:** Binary prediction, risk analysis.
- **Unsuitable Cases:** Data with complex non-linear relationships.

Decision Tree (DT)

- **Description:** An algorithm that uses a tree structure to make decisions based on data features.
- **Advantages:** Easy to interpret, does not require data normalization.
- **Disadvantages:** Prone to overfitting, performance can be poor on imbalanced data.
- **Suitable Cases:** Decision analysis, classification with clear rules.
- **Unsuitable Cases:** Large datasets with many features.

Random Forest (RF)

- **Description:** An ensemble algorithm that combines multiple decision trees to improve accuracy.
- **Advantages:** Reduces overfitting, works well with imbalanced data.
- **Disadvantages:** Difficult to interpret, requires significant computational resources.
- **Suitable Cases:** Complex classification, prediction with imbalanced data.
- **Unsuitable Cases:** Applications requiring clear model interpretation.

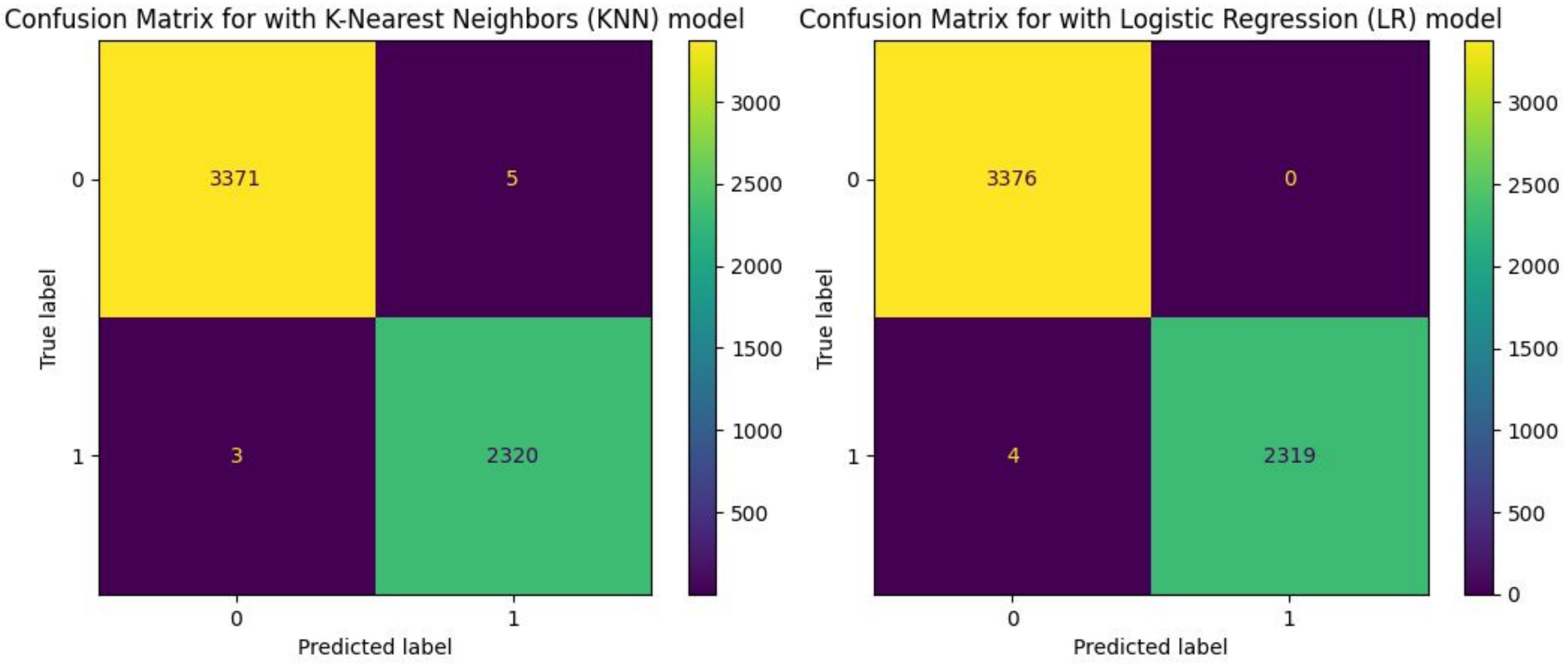
Support Vector Machine (SVM)

- **Description:** An algorithm that finds the best hyperplane to separate classes in the feature space.
- **Advantages:** Effective for high-dimensional data, works well with clear margins.
- **Disadvantages:** Slow for large datasets, requires precise parameter tuning.
- **Suitable Cases:** Text classification, face recognition.
- **Unsuitable Cases:** Large datasets, data with a lot of noise.

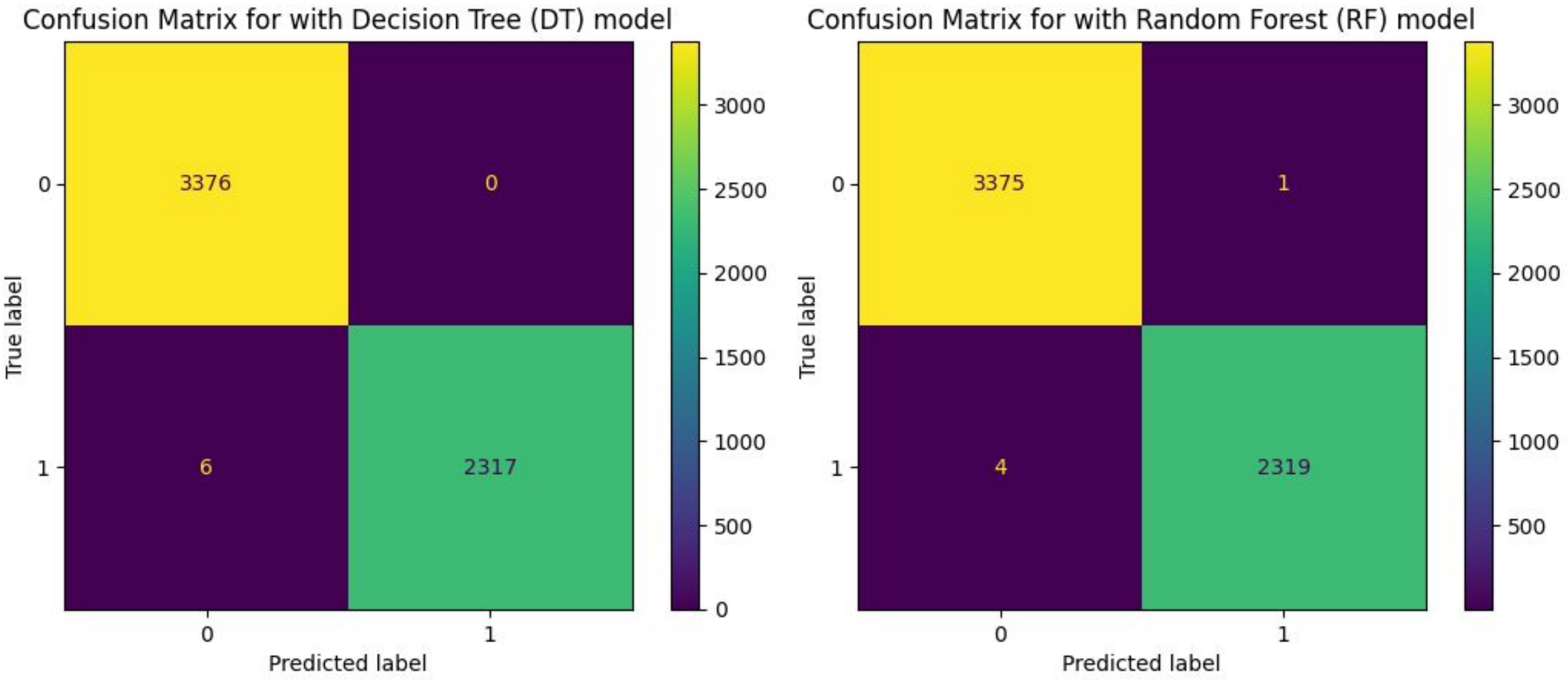
Naive Bayes (NB)

- **Description:** A probabilistic algorithm that uses Bayes' Theorem with the assumption of feature independence.
- **Advantages:** Fast and efficient, works well with categorical data.
- **Disadvantages:** Independence assumption is often unrealistic, performance can be poor with highly correlated data.
- **Suitable Cases:** Text classification, spam detection.
- **Unsuitable Cases:** Data with highly dependent features.

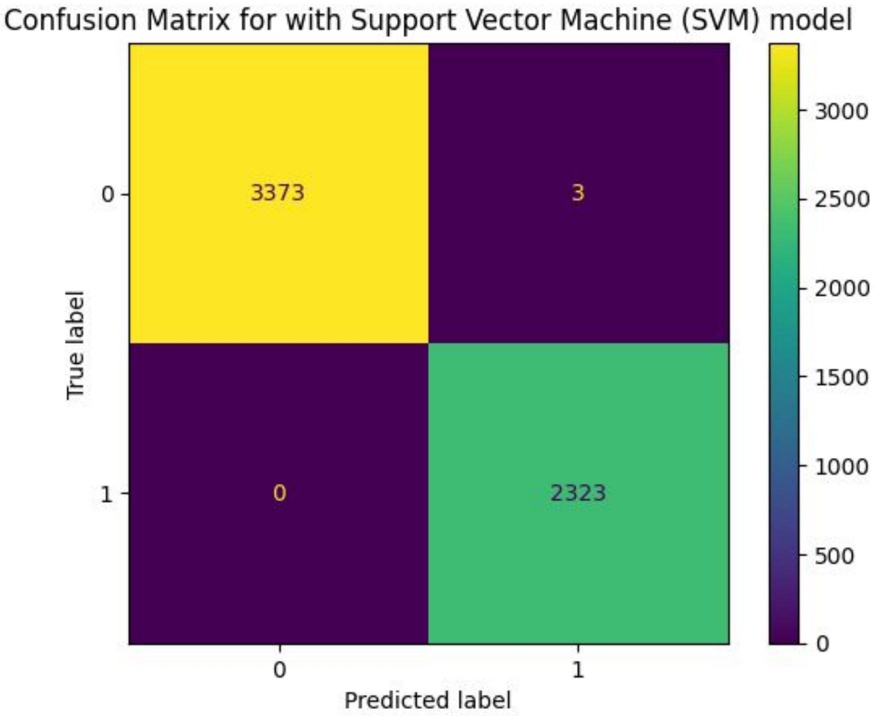
Model Evaluation: Confusion Matrix (1/3)

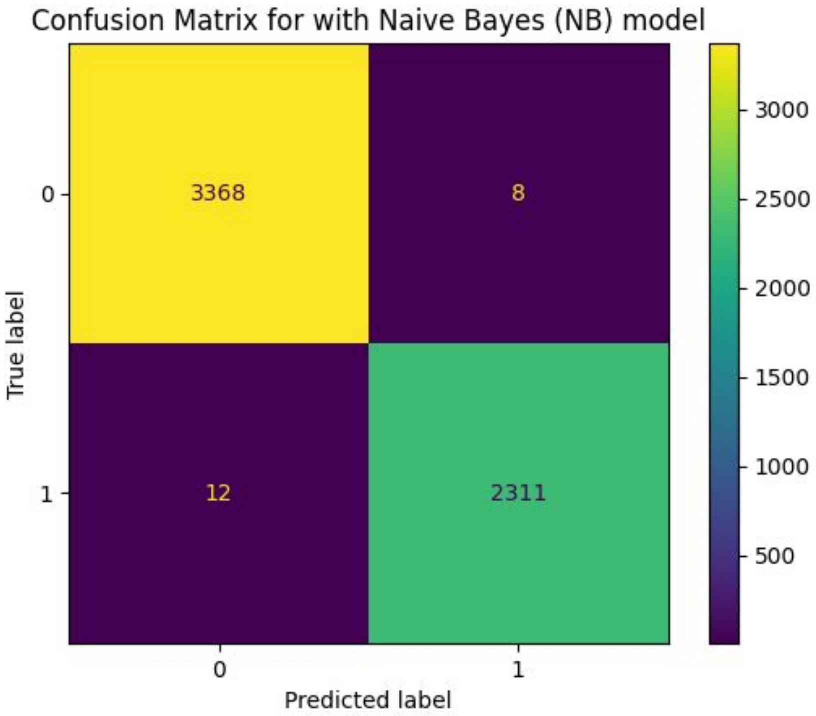


Model Evaluation: Confusion Matrix (2/3)



Model Evaluation: Confusion Matrix (3/3)





Model Evaluation: Performance Metrics

Model	Accuracy	Precision	Recall	F1-Score
K-Nearest Neighbors (KNN)	0.998596	0.997849	0.998709	0.998279
Logistic Regression (LR)	0.999298	1.000000	0.998278	0.999138
Decision Tree (DT)	0.998947	1.000000	0.997417	0.998707
Random Forest (RF)	0.999123	0.999569	0.998278	0.998923
Support Vector Machine (SVM)	0.999474	0.998710	1.000000	0.999355
Naive Bayes (NB)	0.996491	0.996550	0.994834	0.995692

But, why is all of them >90%? Overfitting??

Best models:

Accuracy	Precision	Recall	F1-Score
Support Vector Machine (SVM)	Logistic Regression (LR)	Support Vector Machine (SVM)	Support Vector Machine (SVM)
Logistic Regression (LR)	Decision Tree (DT)	K-Nearest Neighbors (KNN)	Logistic Regression (LR)
Random Forest (RF)	Random Forest (RF)	Logistic Regression (LR)	Random Forest (RF)
Decision Tree (DT)	Support Vector Machine (SVM)	Random Forest (RF)	Decision Tree (DT)
K-Nearest Neighbors (KNN)	K-Nearest Neighbors (KNN)	Decision Tree (DT)	K-Nearest Neighbors (KNN)
Naive Bayes (NB)	Naive Bayes (NB)	Naive Bayes (NB)	Naive Bayes (NB)

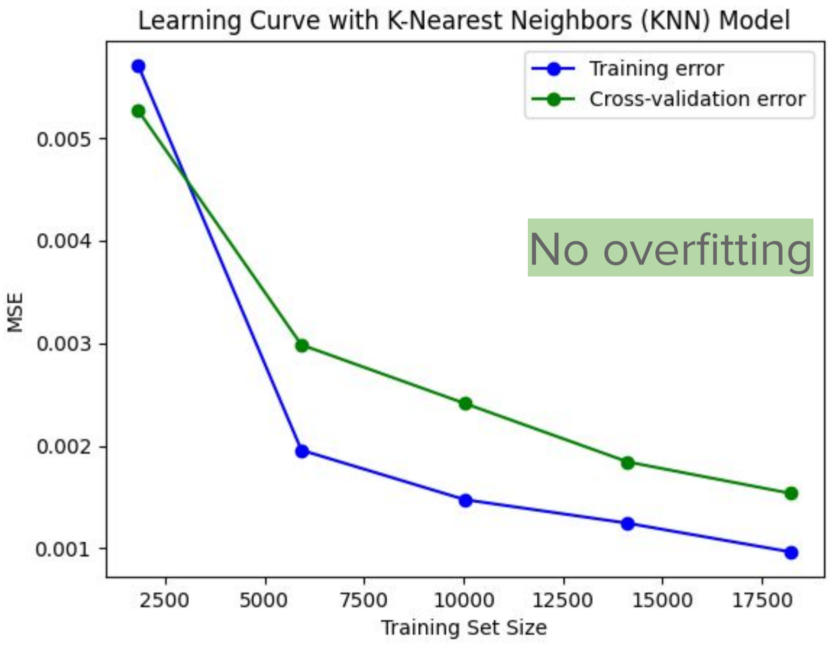
Overfitting Check

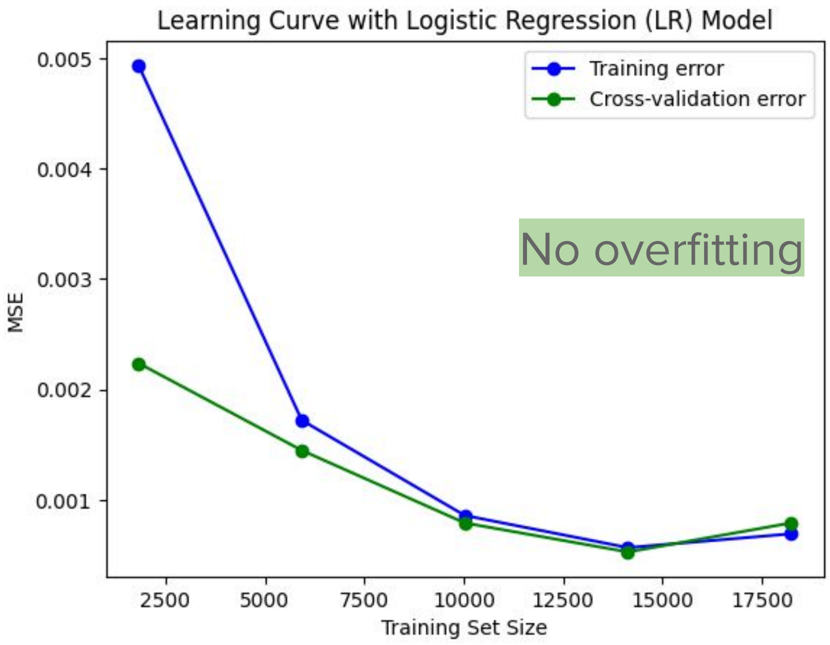
MSE train and MSE test are not that different

Model	Accuracy	Precision	Recall	F1-Score	MSE_train	MSE_test
K-Nearest Neighbors (KNN)	0.998596	0.997849	0.998709	0.998279	0.000834	0.000834
Logistic Regression (LR)	0.999298	1.000000	0.998278	0.999138	0.000395	0.000395
Decision Tree (DT)	0.998947	1.000000	0.997417	0.998707	0.000000	0.000000
Random Forest (RF)	0.999123	0.999569	0.998278	0.998923	0.000000	0.000000
Support Vector Machine (SVM)	0.999474	0.998710	1.000000	0.999355	0.000570	0.000570
Naive Bayes (NB)	0.996491	0.996550	0.994834	0.995692	0.004080	0.004080

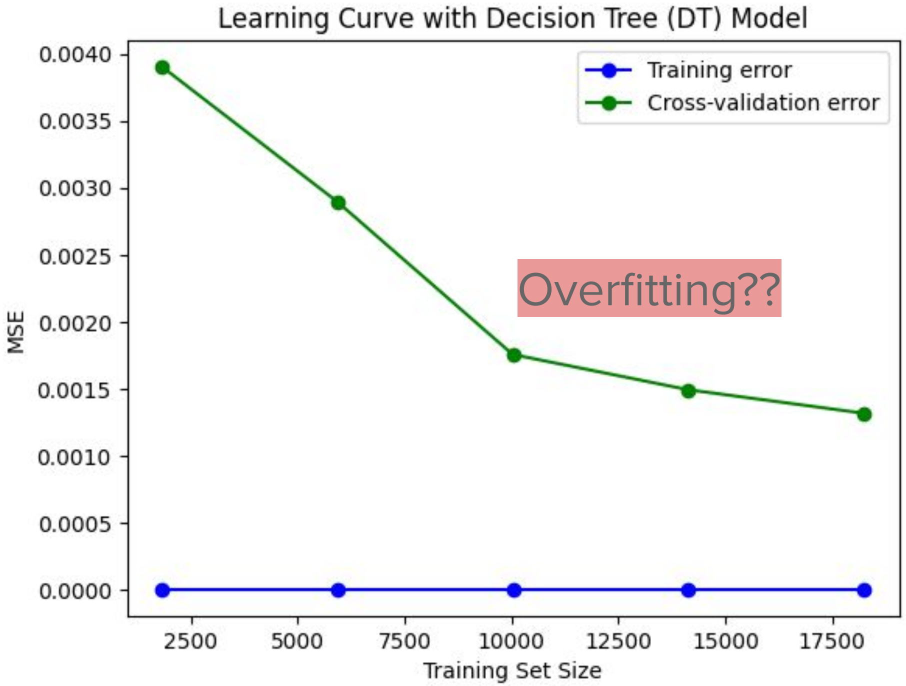
Overfitting Check - Learning Curve (1/3)

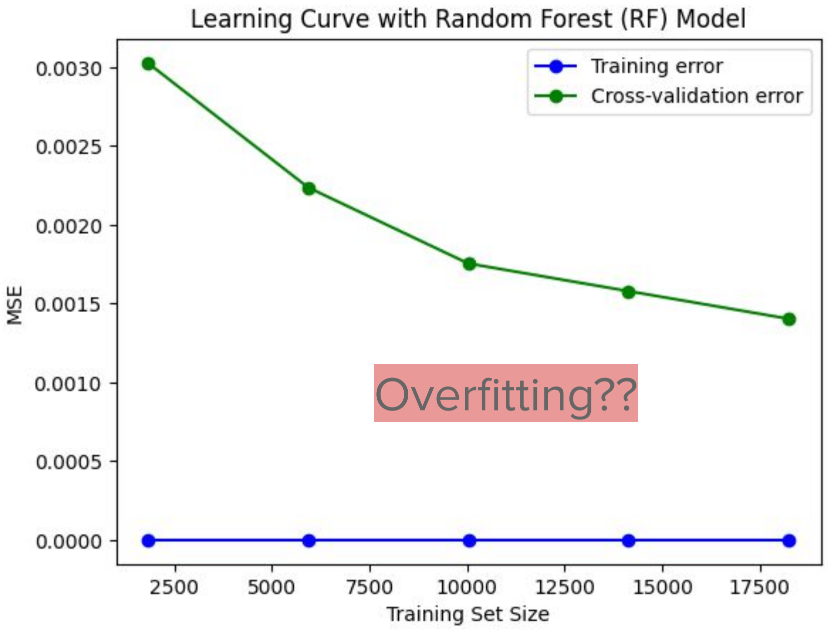
The small gap between training error and cross-validation error suggests that overfitting is not a significant issue.



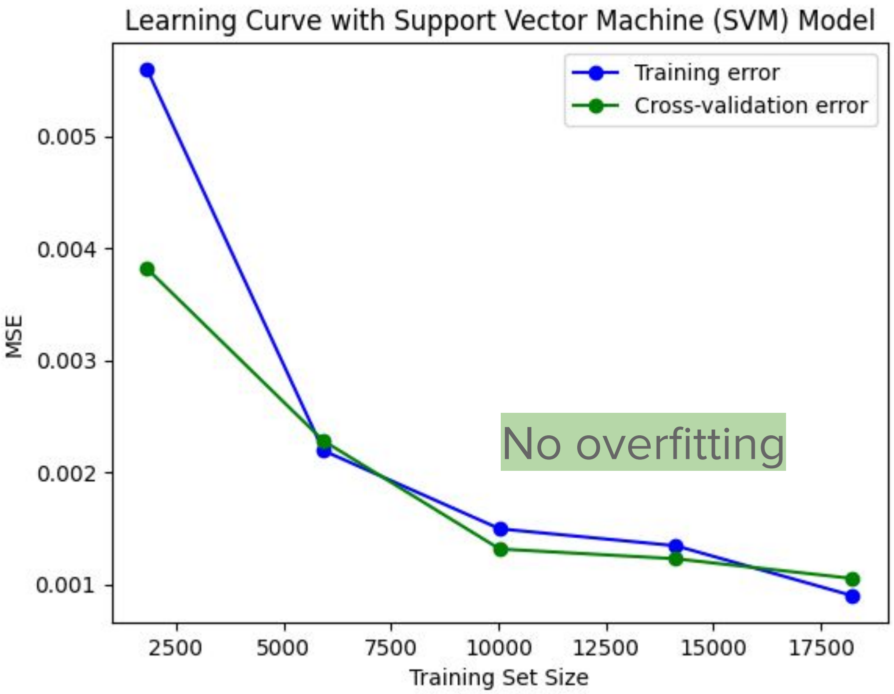


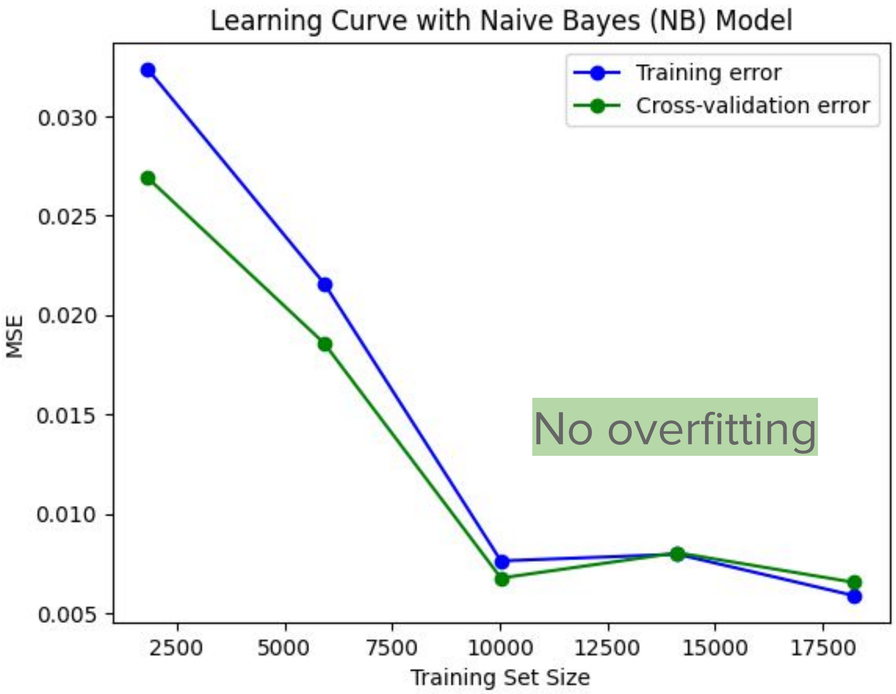
Overfitting Check - Learning Curve (2/3)





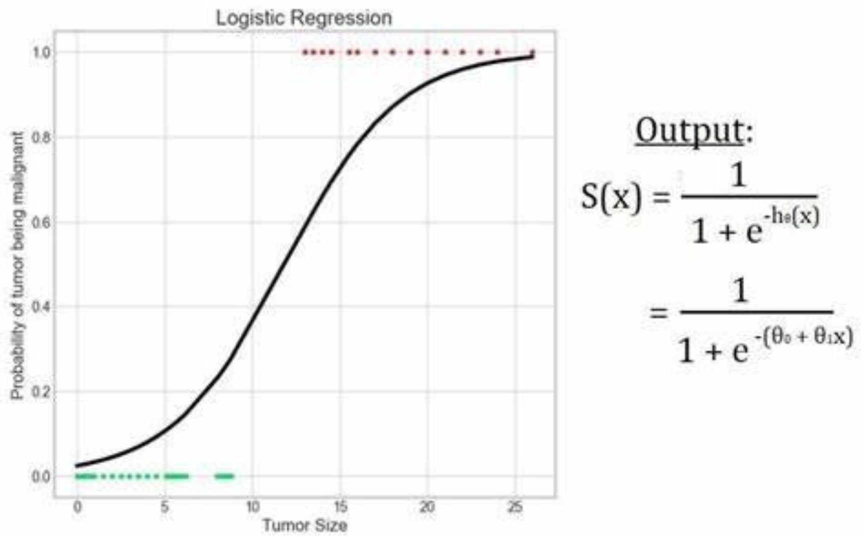
Overfitting Check - Learning Curve (3/3)





I choose Logistic Regression (LR) model...

Among all models, the logistic regression (LR) model provides the best performance metrics and does not indicate overfitting. Additionally, this model also has low complexity.



Hyperparameter Tuning

```
# Define the hyperparameters and their values
param_grid = {
    'C': [0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}

# Perform grid search
def perform_grid_search(param_grid, model, X_train, y_train):
    grid_search = GridSearchCV(model, param_grid, cv=5)
    grid_search.fit(X_train, y_train)
    return grid_search.best_params_
```

	C	penalty	solver
Initial Parameter	1.0	l2	lbfgs
Best Parameter	100.0	l1	liblinear

Re-Evaluate the Model: Confusion Matrix

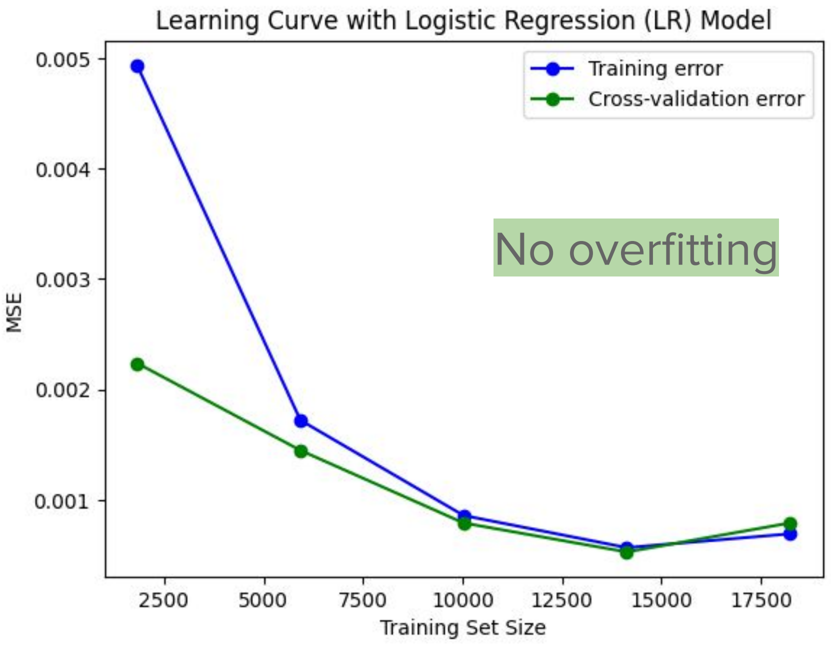
Re-Evaluate the Model: Performance Metrics

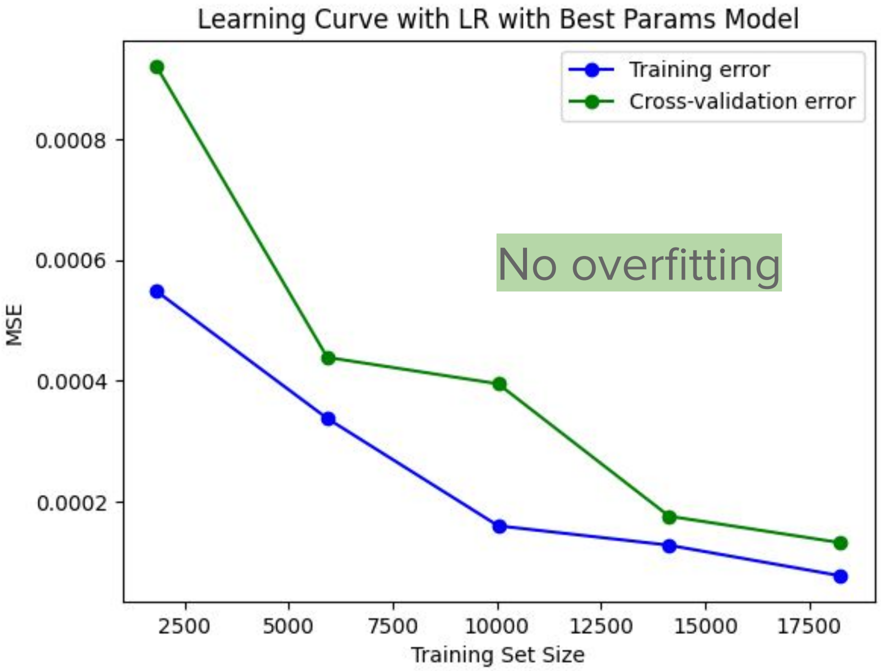
Model	Accuracy	Precision	Recall	F1-Score	MSE_train	MSE_test
Logistic Regression (LR)	0.999298	1.000000	0.998278	0.999138	0.000395	0.000395
LR with Best Params	0.999825	0.999570	1.000000	0.999785	0.000044	0.000044

Better	Worse	Better	Better
Accuracy	**Precision**	**Recall**	**F1-Score**
LR with Best Params	Logistic Regression (LR)	Support Vector Machine (SVM)	LR with Best Params
Support Vector Machine (SVM)	Decision Tree (DT)	LR with Best Params	Support Vector Machine (SVM)
Logistic Regression (LR)	LR with Best Params	K-Nearest Neighbors (KNN)	Logistic Regression (LR)
Random Forest (RF)	Random Forest (RF)	Logistic Regression (LR)	Random Forest (RF)
Decision Tree (DT)	Support Vector Machine (SVM)	Random Forest (RF)	Decision Tree (DT)

Overfitting Check - Learning Curve

The small gap between training error and cross-validation error suggests that overfitting is not a significant issue.





Evaluation Results Before vs After Hyperparameter Tuning

After tuning the hyperparameters of the Logistic Regression model from:

- C: 1, penalty: l2, solver: lbfgs

to:

- C: 100.0, penalty: l1, solver: liblinear,

it was found that

- *accuracy* increased by 0.05%,
- *precision* decreased by 0.04%,
- *recall* increased by 0.17%,
- *F1-score* increased by 0.06%
- False positives: 4 → 0
- False negatives: 0 → 1

Overall, we get a better performance after hyperparameter tuning

Neither model showed indications of overfitting based on MSE of training & test data, as well as the learning curve.