# Take Home Exam

## Task 1

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline


# From the dataset, we can see that it consists 3 columns
# y is the target variable with x1 and x2 as the features
train_data_path = 'synthetic_dataset.csv'
test_data_path = 'synthetic_test_dataset.csv'

train_data = pd.read_csv(train_data_path)
test_data = pd.read_csv(test_data_path)

train_data.head()
```

| | x1 | x2 | y |
|---|---|---|---|
| **0** | 3.989474 | 6.935317 | 1 |
| **1** | -6.248354 | -8.750153 | -1 |
| **2** | -1.419806 | -3.427461 | -1 |
| **3** | -9.266949 | 7.083503 | 1 |
| **4** | -8.487555 | -4.916374 | -1 |

```
test_data.head()
```

| | x1 | x2 | y |
|---|---|---|---|
| **0** | -1.800591 | 2.131077 | -1.0 |
| **1** | -5.116260 | 6.153548 | -1.0 |
| **2** | 1.040485 | 0.889453 | -1.0 |
| **3** | 1.796096 | -1.008297 | -1.0 |
| **4** | -4.039528 | 7.832167 | -1.0 |

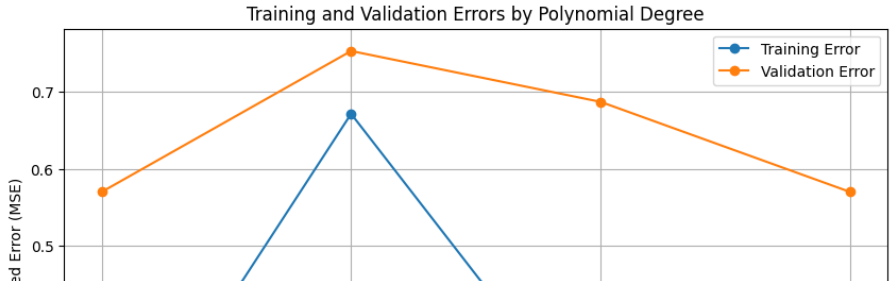
```
# Step 1: Train a linear model and collect the training and validation errors
# Use linear regression algorithm and MSE to get the training and validation errors.

x_train = train_data[['x1', 'x2']]
y_train = train_data['y']
x_test = test_data[['x1', 'x2']]
y_test = test_data['y']

linear_model = LinearRegression()
linear_model.fit(x_train, y_train)

y_train_pred = linear_model.predict(x_train)
y_test_pred = linear_model.predict(x_test)

mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)

print("Training error: ", mse_train)
print("Validation error: ", mse_test)
```

```
Training error:  0.8669392995099426
Validation error:  0.844018025552504
```

```
# Step 2: Train a linear model using quadratic, 3rd order, and 4th order polynomial transforms and collect the training and validation error

# Degrees of the polynomial transformations
degrees = [2, 3, 4]

mse_results = {
    'degree': [],
    'mse_train': [],
    'mse_test': []
}

# Training models for each polynomial degree
for degree in degrees:
    # Create a pipeline that first transforms the data using PolynomialFeatures, then applies LinearRegression
    model = make_pipeline(PolynomialFeatures(degree), LinearRegression())

    model.fit(x_train, y_train)

    y_train_pred = model.predict(x_train)
    y_test_pred = model.predict(x_test)

    mse_train = mean_squared_error(y_train, y_train_pred)
    mse_test = mean_squared_error(y_test, y_test_pred)

    mse_results['degree'].append(degree)
    mse_results['mse_train'].append(mse_train)
    mse_results['mse_test'].append(mse_test)

mse_results_df = pd.DataFrame(mse_results)
mse_results_df
```

| | degree | mse_train | mse_test |
|---|---|---|---|
| **0** | 2 | 0.671256 | 0.752794 |
| **1** | 3 | 0.253624 | 0.686755 |
| **2** | 4 | 0.173705 | 0.569765 |

```
# Step 3: Plot the training and validation errors by order of polynomial transforms, i.e., linear, quadratic, 3rd, and 4th order.
mse_results_df.loc[-1] = [1, mse_train, mse_test]
mse_results_df.index = mse_results_df.index + 1
mse_results_df = mse_results_df.sort_index()

plt.figure(figsize=(10, 6))
plt.plot(mse_results_df['degree'], mse_results_df['mse_train'], label='Training Error', marker='o')
plt.plot(mse_results_df['degree'], mse_results_df['mse_test'], label='Validation Error', marker='o')
plt.xlabel('Degree of Polynomial')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Training and Validation Errors by Polynomial Degree')
plt.xticks(mse_results_df['degree'])
plt.legend()
plt.grid(True)
plt.show()
```




Step 4: Comment on the pattern you observe. Also, comment on which of the models you would pick and why you would pick it.

As the degree of the polynomial increases, both the training and validation MSE decrease, indicating a better fit to the data. However, it's important to be cautious of overfitting, especially with higher degree polynomials, as they tend to fit the training data too closely, potentially at the expense of generalization to new data.

But in order to select the appropriate model, we must depends on balancing the trade-off between underfitting and overfitting. The linear model (1st degree) is likely too simple as indicated by its higher errors. The 4th degree polynomial model, while having the lowest error might be too complex and could overfit the data. Therefore, I would pick 3rd degree polynomial model because it strikes a balance between complexity and the risk of overfitting as evidenced by its significantly lower error than the linear model and only slightly higher error than the 4th degree polynomial.

## Task 2

```
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score


cancer_data_path = 'breast_cancer_dataset_preprocessed.csv'
cancer_data = pd.read_csv(cancer_data_path)
cancer_data.head()
```

| | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | y |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 4.147954 | -4.443183 | -0.068966 | 4.035033 | 0.817574 | -0.476277 | 0.553593 | 1.268819 | M |
| **1** | -4.595154 | -2.684882 | 1.084110 | -0.403925 | 0.410287 | 0.687051 | 0.284184 | 0.260968 | B |
| **2** | -0.755349 | -2.318373 | -1.938275 | 0.279953 | 0.241712 | 3.409801 | 0.092694 | 1.040391 | M |
| **3** | -0.453863 | 0.197572 | -1.037060 | 0.344384 | 0.070598 | -0.822546 | -0.993352 | -0.946259 | B |
| **4** | -3.278680 | -0.792025 | -0.736833 | -1.621295 | -0.085459 | -0.824324 | -0.107042 | -0.291755 | B |

```
X = cancer_data.drop('y', axis=1)
y = cancer_data['y']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=570)

models = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Support Vector Machine': SVC(),
    'K-Nearest Neighbors': KNeighborsClassifier()
}

cv_results = {}

for name, model in models.items():
    cv_result = cross_validate(model, X_train, y_train, cv=5, scoring=['accuracy', 'precision_macro', 'recall_macro', 'f1_macro'])
    cv_results[name] = cv_result

cv_results_mean = {model: {metric: np.mean(scores) for metric, scores in results.items()} for model, results in cv_results.items()}

# Converting to DataFrame for better visualization and sorting by test accuracy
cv_results_mean_df = pd.DataFrame(cv_results_mean).transpose().sort_values(by='test_accuracy', ascending=False)

cv_results_mean_df
```

| | fit_time | score_time | test_accuracy | test_precision_macro | test_recall_macro |
|---|---|---|---|---|---|
| **Logistic Regression** | 0.012789 | 0.014892 | 0.983607 | 0.983543 | 0.981631 |
| **Support Vector Machine** | 0.003481 | 0.006943 | 0.963880 | 0.964082 | 0.958186 |
| **K-Nearest Neighbors** | 0.003018 | 0.011496 | 0.960601 | 0.961701 | 0.954301 |

Since the task is related to binary classification, so I included some popular classification models to find out the best performing model for this dataset.

1. Logistic Regression
2. Decision Tree
3. Random Forest
4. Support Vector Machine (SVM)
5. K-Nearest Neighbors (KNN)

Each model will be first trained on the training set and I used cross-validation to evaluate the performance of each model on the training set to ensure robust assessment.

The models will then be evaluated based on the accuracy, percision, recall and F1-score. The best performing model will be selected based on the cross-validation results and find the model that balances high accuracy with good precision and recall.

Finally, the selected model will be tested on the testing set to estimate the performance of it on new and unseen data.

Based on the cross-validation results above, we can see that the logistic regression model actually shows the best performance among all the models and across all the metrics. It achieves a high level of accuracy while also maintaining a balance between precision and recall. This balance is crucial in medical diagnoses, as both false positives and false negatives have significant implications.

The reason I chose Logistic Regression model is because of:

1. High Accuracy: It correctly classifies the cancer diagnosis most of the time.
2. Balance Between Precision and Recall: Essential in medical diagnostic problems where both false negatives and false positives are critical.
3. Simplicity and Interpretability: Logistic Regression is relatively simple and interpretable compared to more complex models like Random Forest or SVM. This simplicity can be beneficial in a medical context where understanding the decision-making process can be as important as the decision itself.

The performance metrics that I used:

1. Accuracy: Measures the overall correctness of the model.

2. Precision and Recall: Important in scenarios where the cost of false positives and false negatives is high.
3. F1-Score: Harmonic mean of precision and recall, providing a single metric to assess the balance between them.

```
# Final testing of the Logistic Regression model on the test set
logistic_regression_model = LogisticRegression()
logistic_regression_model.fit(X_train, y_train)

y_test_pred = logistic_regression_model.predict(X_test)

test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred, pos_label='M')
test_recall = recall_score(y_test, y_test_pred, pos_label='M')
test_f1_score = f1_score(y_test, y_test_pred, pos_label='M')

test_metrics = {
    'Accuracy': test_accuracy,
    'Precision': test_precision,
    'Recall': test_recall,
    'F1 Score': test_f1_score
}

test_metrics_df = pd.DataFrame([test_metrics], index=['Logistic Regression'])

test_metrics_df
```

| | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| **Logistic Regression** | 0.974026 | 1.0 | 0.933333 | 0.965517 |

From the results above, we can see that the linear regression actually performs very well on the unseen test data. The high accuracy shows that the model is effective in correctly classifying the cancer diagnosis. The perfect precision score indicates that every diagnosis predicted as malignant ('M') by the model is indeed malignant which is crucial in medical diagnostics to avoid false alarms. The recall score is slightly lower but is still consider high which suggesting that the model is also quite good at identifying most of the actual malignant cases. The F1 score which balances precision and recall is also notably high.

In a nutshell, the Logistic Regression model proves to be a robust choice for predicting cancer type based on the provided features with strong