Project1

February 9, 2023

ECE563: AI in Smart Grid

Arturo Galofré (A20521022)

1. **A brief overview of the project**

The field of machine learning has seen tremendous growth in recent years and has led to numerous breakthroughs in a variety of industries. In particular, supervised learning algorithms have been instrumental in solving many real-world problems. This project aims to provide a comprehensive overview of several of the fundamental supervised learning algorithms, including Linear Regression, Logistic Regression, Decision Trees, Support Vector Machines, and k-Nearest Neighbors. By exploring these algorithms, we will gain a deeper understanding of how each of these methods can be used to make predictions based on historical data. We will be working with real-world datasets to demonstrate the strengths and limitations of each algorithm. This project will provide a hands-on approach to learning the basics of supervised machine learning, making it an ideal starting point to begin understanding in this exciting field.

2. **Python code that uses the Scikit-Learn Linear Regression algorithm to predict the number of unit sales given an amount of money spent on radio advertising. You can extract the radio advertising data from the Advertising CSV file as follows (there are other approaches, so do what works for you):**

- `df = pd.read_csv("Advertising.csv")`
- `x_arr = df["radio"].to_numpy()`
- `y_arr = df["sales"].to_numpy()`
- `x_arr = x_arr.reshape(-1,1)`

Since we are only interested in the best fit curve, you may use all the data for training.

3. **Outputs: State the linear model coefficients determined by Linear Regression and the estimated # of units sold given a radio advertising budget of 23 M$ (units used for the radio advertising data).**

[ ]:
```python
# Importing of all necessary libraries for the assingment
import pandas as pd
import numpy as np
import random
from sklearn import datasets
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.datasets import load_diabetes, load_breast_cancer
from sklearn.preprocessing import StandardScaler
```

[ ]:
```
# Money spent on advertising
money_spent = 100

# Load advertising data from csv file
df = pd.read_csv("Advertising.csv")

# Split data into features (x) and target (y)
# .reshape array (rows, columns), -1 tells the number of rows is the number of␣
 ↪elements in the array
# .values converts dataframe to numpy array as skicit only takes data in numpy␣
 ↪array form

x = df["radio"].values.reshape(-1, 1)
y = df["sales"].values

# Create a Linear Regression object
model = LinearRegression()

# Train the model using the training data
# The fit method is used to train a machine learning model on a training␣
 ↪dataset. The fit method takes two required arguments: the first argument is␣
 ↪the training data (also called the "features") and the second argument is␣
 ↪the target data (also called the "labels").
model.fit(x, y)

# Predict the number of unit sales
# .predict predicts an output based on the feature money_spent
radio_spend = [[money_spent]]
radio_spend = np.array(radio_spend)
sales_prediction = model.predict(radio_spend)

# Linear model coefficients
intercept = model.intercept_
coefficient = model.coef_
print("Intercept: ", intercept)
print("Coefficient: ", coefficient)

# Estimation of the number of units sold for 23M$
radio_spend = 23000000
sales_prediction = model.predict([[radio_spend]])
print("Estimated number of units sold for 23M$: ", sales_prediction)
```

```
Intercept:  9.311638095158283
Coefficient:  [0.20249578]
Estimated number of units sold for 23M$:  [4657412.32966421]
```

4. **Observations: How well do your coefficients match the results from our Gradient Descent example? Which method is faster for training the model?**

The coefficients obtained from the scikit-learn implementation of linear regression should match the results from a gradient descent implementation example viewed in class. Both methods are solving for the same linear regression problem, so the coefficients should be the same. After inspecting the graphs presented in class we can observe the "y" intercept is practically the same as the one obtained in the linear reresion model.

As for which method is faster for training the model, scikit-learn uses highly optimized linear algebra libraries so it is generally faster than a custom gradient descent implementation. Gradient descent can be slower compared to linear regression when dealing with large datasets due to the iterative nature of the optimization process.

5. **Python code that uses the Scikit-Learn Logistic Regression algorithm to predict the presence of cancer based on the processed medical image data in the breast cancer dataset. For this part of the project, we will randomly select a subset of features for our classification problem. Use the following code to adjust the seed for the pseudo-random number generator that will randomly select 4 features:**

- `seed = 123456`
- `rng = np.random.default_rng(seed)`
- `idx_feat = (np.floor(30*rng.uniform(size=4))).astype(np.int)`
- `X = cancer["data"][:,idx_feat]`

Again, you may use the entire set of examples in the dataset for training the model. There is no need to set aside a test set for this project.

6. **Outputs: State the seed value, the feature names, and the logistic regression score for the two best and two worst combinations of features found during your testing. Don't worry about finding the optimal solution. We are only interested in gaining some familiarity with the variation in the accuracty of the model.**

[ ]:
```python
# Load the breast cancer dataset
cancer = load_breast_cancer()

# Set the seed value
seed = np.random.randint(100000, 999999, size=10)

# Initialize an empty list to store the results
results = []

# Iterate over 10 combinations of features
for i in range(10):
    # Select 4 features randomly
    rng = np.random.default_rng(seed[i])
    idx_feat = (np.floor(30 * rng.uniform(size=4))).astype(int)
    X = cancer.data[:, idx_feat]
    y = cancer.target

    # Fit a logistic regression model on the selected features
    model = LogisticRegression()

    # Train the model using the training data
    model.fit(X, y)

    # Make predictions on the entire dataset
    y_pred = model.predict(X)

    # Calculate the accuracy score
    score = accuracy_score(y, y_pred)

    # Add the results to the list
    results.append({'features': cancer.feature_names[idx_feat], 'score': score,␣
 ↪'seed': seed[i], 'feature index': idx_feat})

# Convert the results to a Pandas DataFrame
results = pd.DataFrame(results)

# Sort the results by score in ascending order
results = results.sort_values(by='score', ascending=False)

# Print the seed value
print(f"Seed values: {seed}")

# Print the feature names and scores for the two best and two worst combinations
print("\nTwo best combinations of features:")
print(results.head(2)[['features', 'score', 'seed', 'feature index']])

print("\nTwo worst combinations of features:")
print(results.tail(2)[['features', 'score', 'seed', 'feature index']])
```

```
Seed values: [302173 316507 355899 153838 555542 143400 383041 571638 727986
560737]

Two best combinations of features:
                                            features     score    seed  \
1  [mean smoothness, concave points error, worst …  0.926186  316507
7  [perimeter error, mean compactness, worst radi…  0.919156  571638

     feature index
1  [4, 17, 26, 23]
7  [12, 5, 20, 15]
```

```
Two worst combinations of features:
                                            features     score    seed  \
5  [mean symmetry, radius error, perimeter error,…  0.799649  143400
4  [mean compactness, smoothness error, symmetry …  0.690685  555542

    feature index
5  [8, 10, 12, 19]
4  [5, 14, 18, 18]
```

7. **Questions: As you varied the "seed" value, what changes did you notice in the random selection of features? Were any of the features better in terms of increasing the logistic regression score? If you increased the number of features selected, would you get higher scores? What tradeoff would you be making if you selected more features? What other hyperparameters could you tune to improve the scores?**

As the "seed" value changed, the random selection of features also changed, but the underlying dataset remains the same. No single feature was consistently better in terms of increasing the logistic regression score.

Increasing the number of features selected may lead to higher scores, but this would result in the tradeoff of increased complexity and overfitting. Overfitting occurs when a model becomes too complex and memorizes the training data instead of learning the underlying patterns.

To improve the scores, you could tune the hyperparameters such as the regularization strength, which controls the balance between fitting the data and avoiding overfitting, or the solver algorithm used to optimize the parameters of the model.

8. **Python code that uses the Scikit-Learn Decision Tree Regressor algorithm to predict diabetes progression based on a subset of medical features. Similar to the Logistic Regression task above, use the pseudo-random number generator and "seed" to randomly select 4 features for building a decision tree. Use the entire set of examples in the dataset for training the model.**

9. **Outputs: State the seed value, the feature names, the depth of the tree, the number of leaves of the tree and the decision tree coefficient of determination score for the two best and two worst combinations of features found during your testing. Don't worry about finding the optimal solution.**

[ ]:
```python
# Load the diabetes dataset
diabetes = load_diabetes()

# Initialize an empty list to store the results
results = []

# Generate a list of 10 random seeds
seed = np.random.randint(100000, 999999, size=10)

# Loop through each seed
for i in range(10):
    rng = np.random.default_rng(seed[i])
    # Select a subset of features
    idx_feat = (np.floor(10*rng.uniform(size=4))).astype(int)
    X = diabetes.data[:,idx_feat]
    y = diabetes.target

    # Create a DecisionTreeRegressor object
    model = DecisionTreeRegressor()

    # Fit the model on the training data
    model.fit(X, y)

    # Evaluate the model
    score = model.score(X, y)

    # Get the depth and number of leaves of the tree
    depth = model.tree_.max_depth
    leaves = model.tree_.node_count + 1 - model.tree_.n_leaves

    # Add the results to the list
    results.append({'features': [diabetes.feature_names[i] for i in idx_feat],␣
 ↪'depth': depth, 'leaves': leaves, 'score': score, 'seed': seed[i], 'feature␣
 ↪index': idx_feat})

# Convert the results to a Pandas DataFrame
results = pd.DataFrame(results)

# Sort the results by score in ascending order
results = results.sort_values(by='score', ascending=False)

# Print the seed value
print(f"Seed value: {seed}")

# Print the feature names, depth, number of leaves and the decision tree␣
 ↪coefficient of determination score
# for the two best and two worst combinations of features
print("\nTwo best combinations of features:")
print(results.head(2)[['features', 'depth', 'leaves', 'score', 'seed', 'feature␣
 ↪index']])

print("\nTwo worst combinations of features:")
print(results.tail(2)[['features', 'depth', 'leaves', 'score', 'seed', 'feature␣
 ↪index']])
```

```
Seed value: [748845 474403 658597 315089 904593 318856 820800 878963 155233
622306]
```

```
Two best combinations of features:
              features  depth  leaves  score    seed feature index
0  [age, s3, bmi, s1]     20     434    1.0  748845  [0, 6, 2, 4]
1   [s5, s1, s6, age]     15     441    1.0  474403  [8, 4, 9, 0]

Two worst combinations of features:
             features  depth  leaves     score    seed feature index
8  [s5, s1, bmi, s2]     19     435  1.000000  155233  [8, 4, 2, 5]
9  [bp, age, s6, bp]     18     436  0.999993  622306  [3, 0, 9, 3]
```

10. **Questions: As you varied the "seed" value, what changes did you notice in the random selection of features? Were any of the features better in terms of increasing the decision tree score? If you increased the number of features selected, would you get higher scores? What tradeoff would you be making if you selected more features? What other hyperparameters could you tune to improve the scores?**

Varying the "seed" value in the code would change the random selection of features. Different features may be selected, and hence the decision tree score could be different as well. However, increasing the number of features selected would not necessarily result in higher scores. In fact, selecting too many features may lead to overfitting, leading to poor generalization performance on unseen data. The tradeoff of selecting more features would be the risk of overfitting vs the potential benefit of better capturing the underlying relationship between the features and the target.

To improve the scores, one could tune other hyperparameters such as the maximum depth of the tree, the minimum number of samples required to split an internal node, and the minimum number of samples required to be at a leaf node. These hyperparameters can be adjusted to control the complexity of the model and prevent overfitting.

11. **Python code that uses the Scikit-Learn Support Vector Machine algorithm LinearSVC to predict the presence of cancer based on the processed medical image data in the breast cancer dataset. Follow the same procedure as was used for Logistic Regression.**

12. **Outputs: State the seed value, the feature names, and the LinearSVC score for the two best and two worst combinations of features found during your testing. Don't worry about finding the optimal solution.**

[ ]:
```python
# Load the breast cancer dataset
cancer = load_breast_cancer()

# Initialize an empty list to store the results
results = []

# Generate a list of 10 random seeds
seed = np.random.randint(100000, 999999, size=10)

# Loop through each seed
for i in range(10):
    rng = np.random.default_rng(seed[i])
    idx_feat = (np.floor(10*rng.uniform(size=4))).astype(int)
    X = cancer.data[:, idx_feat]
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
    y = cancer.target

    # Create a LinearSVC object
    model = LinearSVC(max_iter=1500)

    # Fit the model on the training data
    model.fit(X, y)

    # Calculate the accuracy score
    score = model.score(X, y)

    # Add the results to the list
    #results.append({'accuracy': score, 'seed': seed[i]})

    # Add the results to the list
    results.append({'features': [cancer.feature_names[i] for i in idx_feat],
                    'score': score, 'seed': seed[i], 'feature index': idx_feat})

# Convert the results to a Pandas DataFrame
results = pd.DataFrame(results)

# Sort the results by accuracy in ascending order
results = results.sort_values(by='score', ascending=False)

# Print the seed value
print(f"Seed value: {seed}")

# Print the accuracy score for the two best and two worst seeds
print("\nTwo best seeds:")
print(results.head(2)[['features', 'score', 'seed']])

print("\nTwo worst seeds:")
print(results.tail(2)[['features', 'score', 'seed']])
```

```
Seed value: [634638 782590 749754 467201 947057 478985 199746 995979 412313
468623]

Two best seeds:
                                            features     score    seed
1  [mean concave points, mean compactness, mean t…  0.938489  782590
6  [mean radius, mean compactness, mean smoothnes…  0.933216  199746

Two worst seeds:
                                            features     score    seed
```

```
5  [mean fractal dimension, mean perimeter, mean …  0.905097  478985
0  [mean texture, mean texture, mean concavity, m…  0.887522  634638
```

13. **Questions: As you varied the “seed” value, what changes did you notice in the random selection of features? Were any of the features better in terms of increasing the LinearSVC score? If you increased the number of features selected, would you get higher scores? What tradeoff would you be making if you selected more features? What other hyperparameters could you tune to improve the scores?**

As the “seed” value changes, the random selection of features will change. This means that each time you run the code with a different seed value, you will get a different set of features selected. It is not possible to determine if any of the features are better in terms of increasing the LinearSVC score without running the model for every combination of features and for multiple values of the seed. This would require a lot of computation time.

If you increase the number of features selected, you might get higher scores, but there is no guarantee. Selecting more features can increase the complexity of the model and lead to overfitting, where the model performs well on the training data but poorly on unseen data. In addition, by selecting more features, you would be making a tradeoff between model complexity and generalization performance. In general, a more complex model is more likely to overfit the data, while a simpler model is more likely to underfit.

There are several hyperparameters that could be tuned to improve the scores, such as the regularization strength (applying a penalty to increasing the magnitude of parameter values in order to reduce overfitting) and the choice of the loss function. The regularization strength controls the tradeoff between accuracy and model complexity, while the choice of loss function determines the criteria used for optimization.

To conclude, in the development of the code, problems arose as the model did not converge, to solve this, we opted to normalize the data and increse the number of iterations which seemed to solve the convergence error.

14. **Python code that uses the Scikit-Learn KNeighborsRegressor algorithm to predict diabetes progression based on a subset of medical features. Follow the same procedure as was used for the Decision Tree Regressor.**

15. **Outputs: State the seed value, the feature names, the number of neighbors and the KNeighborsRegressor coefficient of determination score for the two best and two worst combinations of features found during your testing. Don’t worry about finding the optimal solution.**

[ ]:
```python
# Load the diabetes dataset
diabetes = load_diabetes()

# Initialize an empty list to store the results
results = []

# Generate a list of 10 random seeds
seed = np.random.randint(100000, 999999, size=10)

# Loop through each seed
```

```
for i in range(10):
    rng = np.random.default_rng(seed[i])
    # Select a subset of features
    idx_feat = (np.floor(10*rng.uniform(size=4))).astype(int)
    X = diabetes.data[:, idx_feat]
    y = diabetes.target

    # Create a KNeighborsRegressor object
    model = KNeighborsRegressor()

    # Fit the model on the training data
    model.fit(X, y)

    # Evaluate the model
    score = model.score(X, y)

    # Add the results to the list
    results.append({'features': [diabetes.feature_names[i] for i in idx_feat],
                    'score': score, 'seed': seed[i], 'feature index': idx_feat,␣
 ↪'neighbors':np.size(model.kneighbors(return_distance=False),1)})

# Convert the results to a Pandas DataFrame
results = pd.DataFrame(results)

# Sort the results by score in ascending order
results = results.sort_values(by='score', ascending=False)

# Print the seed value
print(f"Seed value: {seed}")

# Print the feature names, depth, number of leaves and the decision tree␣
 ↪coefficient of determination score
# for the two best and two worst combinations of features
print("\nTwo best combinations of features:")
print(results.head(2)[['features', 'score', 'seed', 'feature index',␣
 ↪'neighbors']])

print("\nTwo worst combinations of features:")
print(results.tail(2)[['features', 'score', 'seed', 'feature index',␣
 ↪'neighbors']])
```

```
Seed value: [893914 833534 252137 827452 102802 314454 721234 801847 206437
970446]

Two best combinations of features:
            features     score    seed feature index  neighbors
8  [bp, bmi, s6, s5]  0.576376  206437  [3, 2, 9, 8]          5
```

```
1  [s2, bmi, s4, s6]  0.556789  833534  [5, 2, 7, 9]          5

Two worst combinations of features:
             features     score    seed feature index  neighbors
7  [bmi, bmi, s2, s1]  0.481650  801847  [2, 2, 5, 4]          5
6    [s5, bp, s2, bp]  0.457466  721234  [8, 3, 5, 3]          5
```

16. **Questions: As you varied the "seed" value, what changes did you notice in the random selection of features? Were any of the features better in terms of increasing the k-Nearest Neighbors score? If you increased the number of features selected, would you get higher scores? What tradeoff would you be making if you selected more features? What other hyperparameters could you tune to improve the scores?**

As you vary the "seed" value, the random selection of features will change. This is because the random number generator is being initialized with a different seed each time, resulting in a different sequence of random numbers being generated.

In terms of increasing the k-Nearest Neighbors score, it is not possible to determine which features are better based on a single random selection of features. It would be necessary to run multiple trials with different seed values and different combinations of features to determine which combinations of features result in higher scores.

Increasing the number of features selected may result in higher scores, but this comes with a tradeoff. Selecting more features can lead to overfitting, where the model becomes too complex and starts to memorize the training data instead of generalizing to new data.

There are a number of other hyperparameters that can be tuned to improve the scores, including the number of neighbors (k), the distance metric used to calculate the distances between examples, and the weighting scheme used to assign weights to the neighbors in making predictions.

17. **A brief conclusion for the project.**

In this project, we explored several of the fundamental supervised learning algorithms, including Linear Regression, Logistic Regression, Decision Trees, Support Vector Machines, and k-Nearest Neighbors. Through this exploration, we learned about the strengths and weaknesses of each of these algorithms and how they can be used to make predictions based on a set of input features.

In terms of future assignments, this knowledge will be incredibly useful. Many data science projects require the use of machine learning algorithms, and having a solid understanding of these algorithms will allow us to select the most appropriate one for the task at hand. Additionally, having hands-on experience with these algorithms will allow us to implement them quickly and effectively in future projects.

Overall, this project was a valuable learning experience that has given us a deeper understanding of supervised learning algorithms. With this knowledge, we are well-equipped to tackle more complex data science problems in the future.