ml-gwp03

July 16, 2024

[ ]:
```
# load libraries
import matplotlib.pyplot as plt
import numpy as np
import math
import pandas as pd
from scipy.optimize import brute, fmin
from scipy.integrate import quad
import yfinance as yf
import pandas_datareader as pdr  # Access FRED
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
```

1 Issue 1: Optimizing hyperparameters

** Code exaple for ...**

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc, confusion_matrix
```

[ ]:
```
#Loading the Credit Card Default dataset
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00350/
↪default%20of%20credit%20card%20clients.xls'
df = pd.read_excel(url, header=1)

features = df.columns[1:-1]
```

```
X = df[features]
y = df['default payment next month']
df

#Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
```

[ ]:
```
svm = SVC()
param_grid = {
    'svm__C': [0.1, 1, 10, 100],
    'svm__kernel': ['linear', 'rbf', 'poly'],
    'svm__gamma': ['scale', 'auto']
}
```

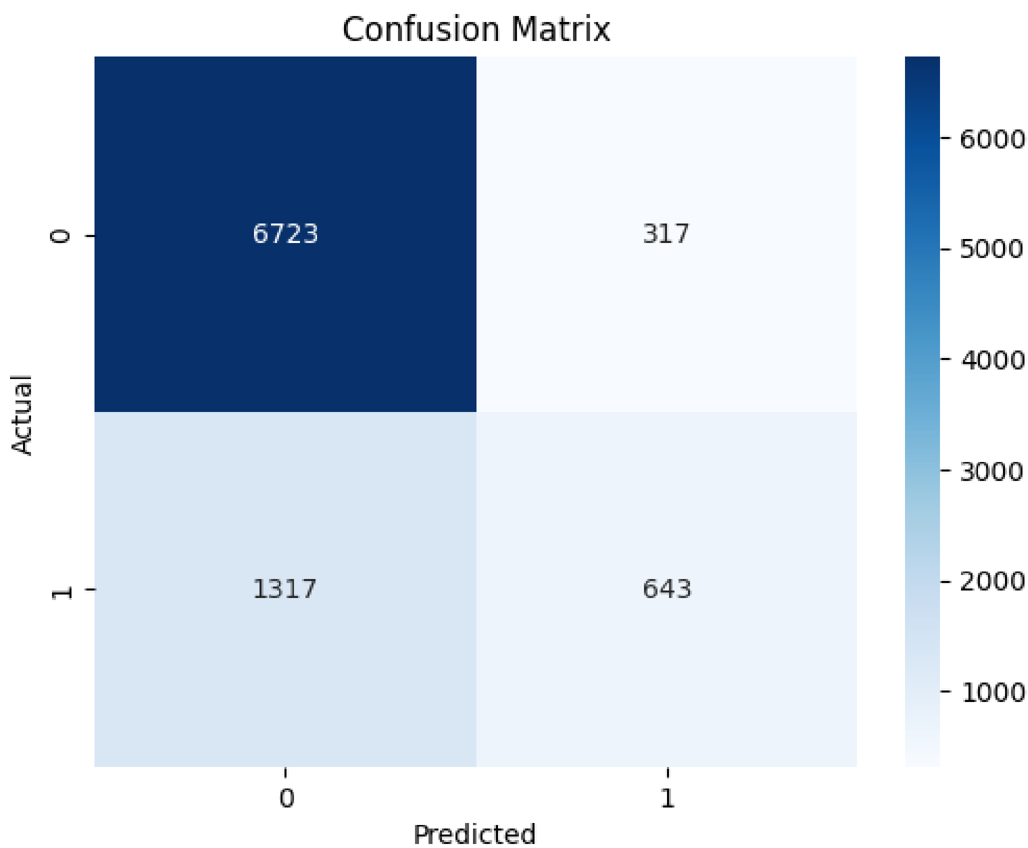
[ ]:
```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svm', svm)
])

#Training the model
pipeline.fit(X_train, y_train)
```

[ ]:
```
Pipeline(steps=[('scaler', StandardScaler()), ('svm', SVC())])
```

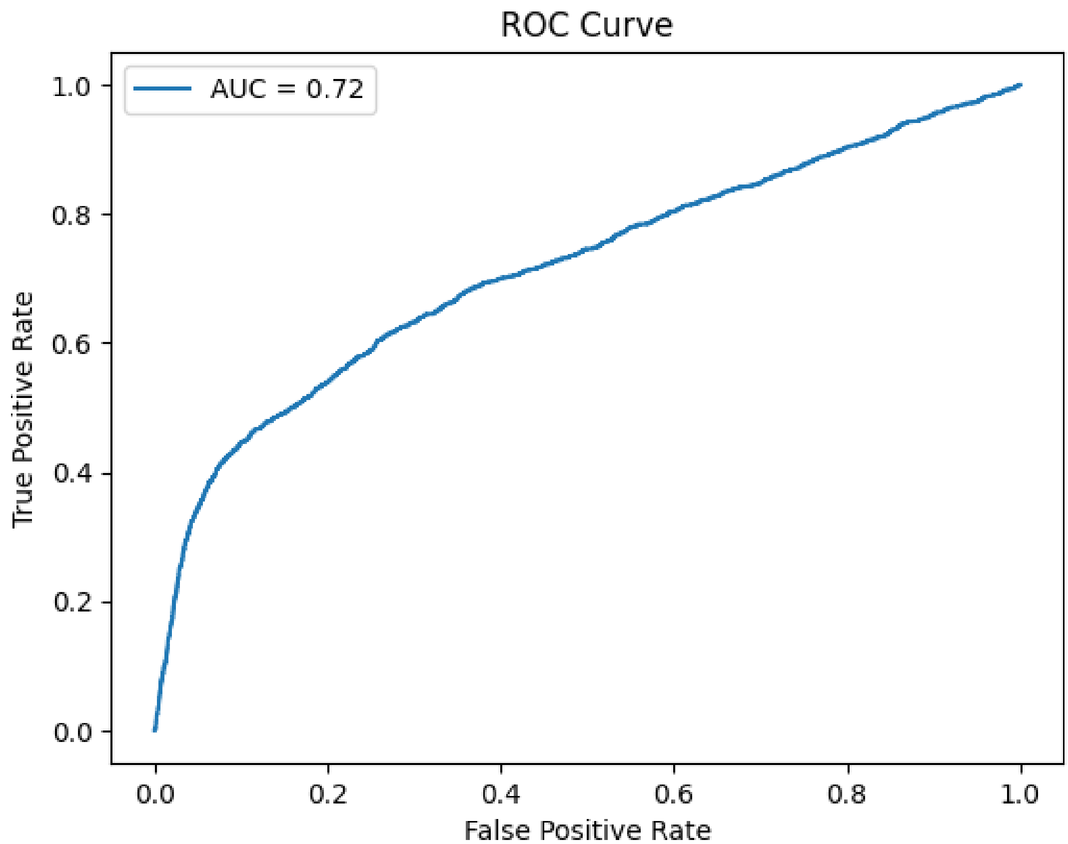
[ ]:
```
y_pred = pipeline.predict(X_test)

#Evaluation of the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(report)
```

```
Accuracy: 0.8184444444444444
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      7040
           1       0.67      0.33      0.44      1960

    accuracy                           0.82      9000
   macro avg       0.75      0.64      0.67      9000
weighted avg       0.80      0.82      0.79      9000
```

[ ]:
```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



[ ]:
```
y_prob = pipeline.decision_function(X_test)
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.2f}')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



[ ]:

2 Issue 2: Optimizing the Bias-Variance Tradeoff

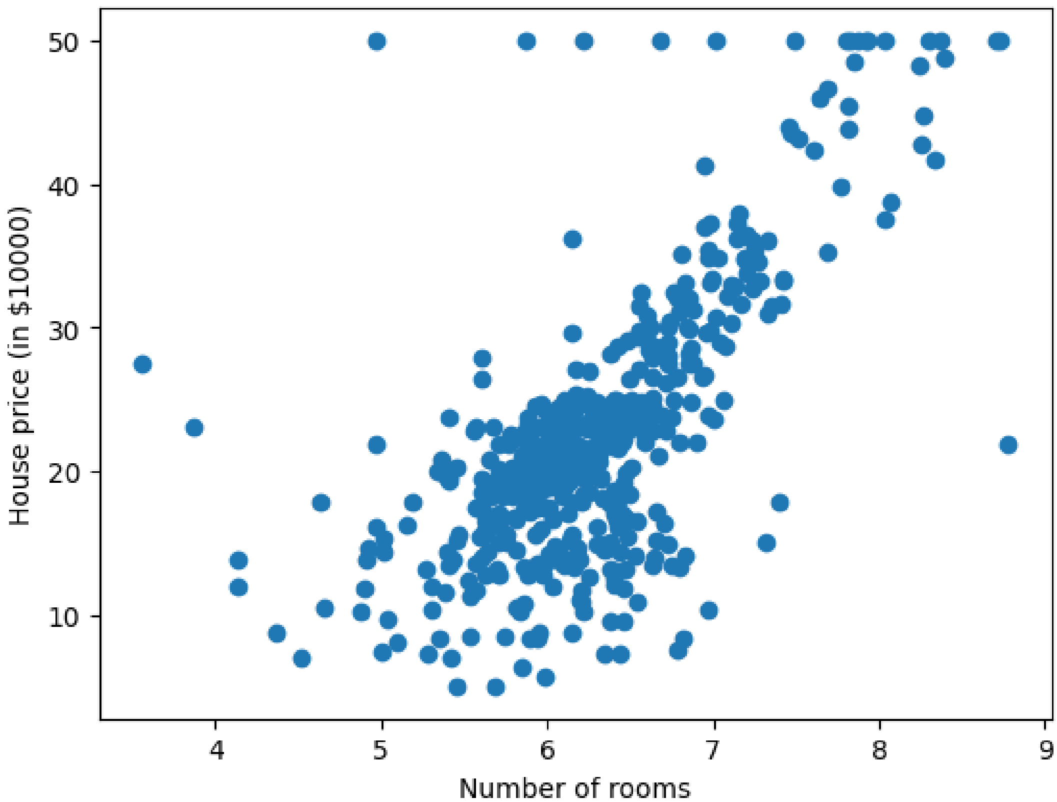
[ ]:

```
# estimate the bias and variance for a regression model
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from mlxtend.evaluate import bias_variance_decomp

# load dataset: boston housing (https://github.com/jbrownlee/Datasets/blob/
 ↪master/housing.names)
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
dataframe = read_csv(url, header=None)
# separate into inputs and outputs
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
```

```
X = X[:,5].reshape(-1, 1)
# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=1)


from sklearn import svm
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
import numpy as np
import matplotlib.pyplot as plt

degrees = [0,1,2,3,4]

mse_all = []
bias_all = []
var_all = []
for d in degrees:
    model = make_pipeline(PolynomialFeatures(d), LinearRegression())

#clf = svm.SVR(kernel='linear')

    mse, bias, var = bias_variance_decomp(model, X_train, y_train, X_test,␣
 ↪y_test, loss='mse', num_rounds=200, random_seed=1)

    mse_all.append(mse)
    bias_all.append(bias)
    var_all.append(var)

mse_all = np.array(mse_all)
bias_all = np.array(bias_all)
var_all = np.array(var_all)
```

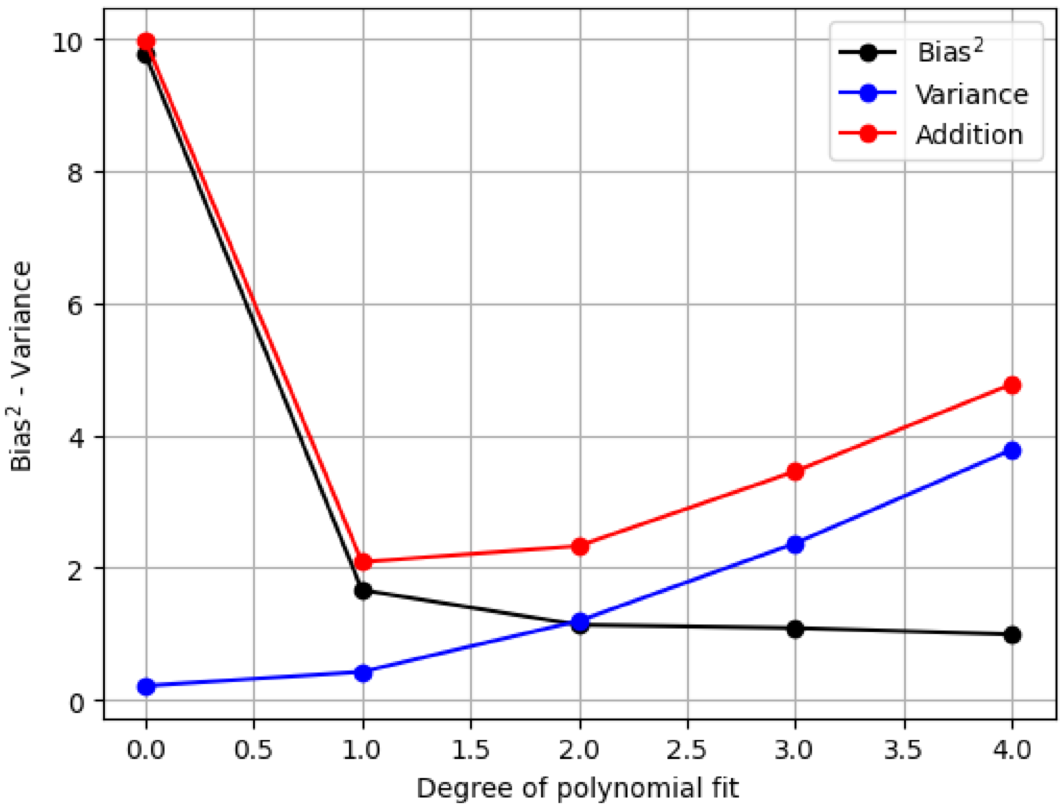
[ ]:
```
plt.scatter(X, y)
plt.xlabel('Number of rooms')
plt.ylabel('House price (in $10000)')
```

[ ]: Text(0, 0.5, 'House price (in $10000)')



[ ]:
```
plt.figure()
plt.plot(degrees, bias_all**2/1000, '-ok', label=r'Bias$^2$')
plt.plot(degrees, var_all, '-ob', label=r'Variance')
plt.plot(degrees, bias_all**2/1000+var_all, '-or', label=r'Addition')
plt.xlabel('Degree of polynomial fit')
plt.ylabel(r'Bias$^2$ - Variance')
plt.grid()
plt.legend(loc='best')
#plt.ylim(0,1e5)
```

[ ]: <matplotlib.legend.Legend at 0x78097bdc6fb0>



3 Issue 3: Applying Ensemble Learning- Bagging, Boosting or Stacking

[ ]:
```
# Define the tickers
tickers = {
    "S&P 500": "^GSPC",
    "Gold": "GC=F",
    "Bitcoin": "BTC-USD",
    "Silver": "SI=F",
    "EURUSD": "EURUSD=X"
}

# Define the period and interval
period = "5y"   # 10 year of data
interval = "1d"   # daily data

# Download the data
data = {}
for name, ticker in tickers.items():
    data[name] = yf.download(ticker, period=period, interval=interval)
```

```python
# Combine the data into a single DataFrame
combined_data = pd.DataFrame({
    "Bitcoin": data["Bitcoin"]["Close"],
    "S&P 500": data["S&P 500"]["Close"],
    "Gold": data["Gold"]["Close"],
    "Silver": data["Silver"]["Close"],
    "EURUSD": data["EURUSD"]["Close"]
})

# Drop rows with missing values
combined_data.dropna(inplace=True)

# Display the first few rows of the combined DataFrame
combined_data.head()
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

[ ]:

	Bitcoin	S&P 500	Gold	Silver	EURUSD
Date					
2019-07-16	9477.641602	3004.040039	1409.199951	15.600000	1.126177
2019-07-17	9693.802734	2984.419922	1421.300049	15.893000	1.121227
2019-07-18	10666.482422	2995.110107	1426.099976	16.120001	1.122965
2019-07-19	10530.732422	2976.610107	1425.099976	16.117001	1.126152
2019-07-22	10343.106445	2985.030029	1425.300049	16.340000	1.121831

[ ]:
```python
### Separate Dataset into two, one is testing dataset, the other is the
 ↪training dataset
train_data, test_data = train_test_split(combined_data, test_size=0.2,
 ↪shuffle=False)

print("The training dataset is:", len(train_data), ", and the testing dataset
 ↪is:",len(test_data))
```

```
The training dataset is: 1004 , and the testing dataset is: 252
```

[ ]:
```python
# Use S&P 500, Gold, Silver and EURUSD to predict the Bitcoin price
# Separate features and target variable
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers import SGD
```

```python
X_train = train_data.drop("Bitcoin", axis=1)
y_train = train_data["Bitcoin"]
X_test = test_data.drop("Bitcoin", axis=1)
y_test = test_data["Bitcoin"]

def create_model_machine_learning():
    inputs = Input(shape=(X_train.shape[1],))
    x = Dense(64, activation='relu')(inputs)
    x = Dense(64, activation='relu')(x)
    outputs = Dense(1)(x)
    model = Model(inputs, outputs)
    model.compile(optimizer=Adam(), loss='mean_squared_error', metrics=['mae'])
    return model

# Define model creation function
def create_model_neural_network():
    tf.random.set_seed(46)  # Set random seed for reproducibility
    model = Sequential([Dense(1, input_shape=(X_train.shape[1],))])
    model.compile(
        loss='mae',  # Mean Absolute Error (MAE) as the loss function
        optimizer=SGD(learning_rate=0.01, momentum=0.9),  # SGD optimizer with␣
 ↪momentum
        metrics=['mae']  # Performance metric is MAE
    )
    return model

# Instantiate models
models = [create_model_machine_learning(),create_model_neural_network()]
```

[ ]:
```python
# Train models
for model in models:
    model.fit(X_train, y_train, epochs=50, batch_size=16, validation_split=0.2,␣
 ↪verbose=1)

# Evaluate individual models
for i, model in enumerate(models):
    score = model.evaluate(X_test, y_test, verbose=0)
    print(f'Model Test MAE: {score[1]:.4f}')  # Access the MAE metric directly

# Ensemble prediction by averaging
predictions = [model.predict(X_test) for model in models]
ensemble_predictions = np.mean(predictions, axis=0)
ensemble_mae = np.mean(np.abs(ensemble_predictions - y_test.values.reshape(-1,␣
 ↪1)))

print(f'Ensemble Test MAE: {ensemble_mae:.4f}')
```

```
Epoch 1/50
51/51 [==============================] - 0s 4ms/step - loss: 96951000.0000 -
mae: 7664.8589 - val_loss: 96318736.0000 - val_mae: 8426.9229
Epoch 2/50
51/51 [==============================] - 0s 3ms/step - loss: 96411104.0000 -
mae: 7609.5435 - val_loss: 89521768.0000 - val_mae: 8147.3271
Epoch 3/50
51/51 [==============================] - 0s 3ms/step - loss: 95980264.0000 -
mae: 7620.2012 - val_loss: 66947516.0000 - val_mae: 7130.6748
Epoch 4/50
51/51 [==============================] - 0s 4ms/step - loss: 95604464.0000 -
mae: 7607.7622 - val_loss: 109640064.0000 - val_mae: 8980.3965
Epoch 5/50
51/51 [==============================] - 0s 3ms/step - loss: 96690464.0000 -
mae: 7664.9756 - val_loss: 77989088.0000 - val_mae: 7650.7480
Epoch 6/50
51/51 [==============================] - 0s 3ms/step - loss: 95742824.0000 -
mae: 7628.5244 - val_loss: 101820352.0000 - val_mae: 8654.7617
Epoch 7/50
51/51 [==============================] - 0s 3ms/step - loss: 96140496.0000 -
mae: 7611.7354 - val_loss: 80973160.0000 - val_mae: 7784.4150
Epoch 8/50
51/51 [==============================] - 0s 4ms/step - loss: 95378008.0000 -
mae: 7602.7231 - val_loss: 74098400.0000 - val_mae: 7478.0337
Epoch 9/50
51/51 [==============================] - 0s 3ms/step - loss: 95907936.0000 -
mae: 7611.8145 - val_loss: 97320024.0000 - val_mae: 8476.6992
Epoch 10/50
51/51 [==============================] - 0s 4ms/step - loss: 96881360.0000 -
mae: 7649.2896 - val_loss: 83152616.0000 - val_mae: 7879.0981
Epoch 11/50
51/51 [==============================] - 0s 3ms/step - loss: 96341336.0000 -
mae: 7627.1943 - val_loss: 106192496.0000 - val_mae: 8836.5068
Epoch 12/50
51/51 [==============================] - 0s 3ms/step - loss: 95673992.0000 -
mae: 7584.9473 - val_loss: 89127480.0000 - val_mae: 8139.7935
Epoch 13/50
51/51 [==============================] - 0s 3ms/step - loss: 95349128.0000 -
mae: 7584.5952 - val_loss: 117971088.0000 - val_mae: 9318.1240
Epoch 14/50
51/51 [==============================] - 0s 3ms/step - loss: 97305312.0000 -
mae: 7648.9438 - val_loss: 85364528.0000 - val_mae: 7979.9463
Epoch 15/50
51/51 [==============================] - 0s 3ms/step - loss: 95762192.0000 -
mae: 7573.6943 - val_loss: 69228512.0000 - val_mae: 7247.5439
Epoch 16/50
51/51 [==============================] - 0s 4ms/step - loss: 95832016.0000 -
mae: 7568.7090 - val_loss: 77108072.0000 - val_mae: 7619.5317
```

```
Epoch 17/50
51/51 [==============================] - 0s 3ms/step - loss: 95967200.0000 -
mae: 7578.4297 - val_loss: 81978920.0000 - val_mae: 7835.7363
Epoch 18/50
51/51 [==============================] - 0s 3ms/step - loss: 95407688.0000 -
mae: 7558.5967 - val_loss: 76039152.0000 - val_mae: 7572.5005
Epoch 19/50
51/51 [==============================] - 0s 3ms/step - loss: 95322664.0000 -
mae: 7553.5098 - val_loss: 87747608.0000 - val_mae: 8085.9087
Epoch 20/50
51/51 [==============================] - 0s 4ms/step - loss: 94490728.0000 -
mae: 7523.1543 - val_loss: 103334296.0000 - val_mae: 8723.7256
Epoch 21/50
51/51 [==============================] - 0s 3ms/step - loss: 95047464.0000 -
mae: 7526.6592 - val_loss: 107083352.0000 - val_mae: 8875.8760
Epoch 22/50
51/51 [==============================] - 0s 3ms/step - loss: 95718136.0000 -
mae: 7609.8213 - val_loss: 87069816.0000 - val_mae: 8060.8711
Epoch 23/50
51/51 [==============================] - 0s 3ms/step - loss: 94977664.0000 -
mae: 7541.2173 - val_loss: 97948328.0000 - val_mae: 8511.9551
Epoch 24/50
51/51 [==============================] - 0s 3ms/step - loss: 94433088.0000 -
mae: 7530.3071 - val_loss: 76552648.0000 - val_mae: 7603.7188
Epoch 25/50
51/51 [==============================] - 0s 3ms/step - loss: 94312640.0000 -
mae: 7524.7627 - val_loss: 76822928.0000 - val_mae: 7615.6846
Epoch 26/50
51/51 [==============================] - 0s 3ms/step - loss: 93966328.0000 -
mae: 7506.3911 - val_loss: 113306168.0000 - val_mae: 9129.1787
Epoch 27/50
51/51 [==============================] - 0s 4ms/step - loss: 95669576.0000 -
mae: 7579.0991 - val_loss: 77592304.0000 - val_mae: 7649.6538
Epoch 28/50
51/51 [==============================] - 0s 4ms/step - loss: 93644904.0000 -
mae: 7475.7017 - val_loss: 111876464.0000 - val_mae: 9072.3623
Epoch 29/50
51/51 [==============================] - 0s 3ms/step - loss: 95924032.0000 -
mae: 7624.7290 - val_loss: 72838664.0000 - val_mae: 7434.2085
Epoch 30/50
51/51 [==============================] - 0s 3ms/step - loss: 94767168.0000 -
mae: 7531.5435 - val_loss: 85014144.0000 - val_mae: 7977.3252
Epoch 31/50
51/51 [==============================] - 0s 3ms/step - loss: 94030376.0000 -
mae: 7509.6226 - val_loss: 67936736.0000 - val_mae: 7192.6113
Epoch 32/50
51/51 [==============================] - 0s 3ms/step - loss: 94798928.0000 -
mae: 7547.5854 - val_loss: 80463600.0000 - val_mae: 7782.5620
```

```
Epoch 33/50
51/51 [==============================] - 0s 3ms/step - loss: 95120504.0000 -
mae: 7610.8618 - val_loss: 64886872.0000 - val_mae: 7027.0596
Epoch 34/50
51/51 [==============================] - 0s 3ms/step - loss: 93970024.0000 -
mae: 7496.5063 - val_loss: 94866608.0000 - val_mae: 8389.7285
Epoch 35/50
51/51 [==============================] - 0s 3ms/step - loss: 94136336.0000 -
mae: 7528.8760 - val_loss: 73422112.0000 - val_mae: 7461.1128
Epoch 36/50
51/51 [==============================] - 0s 3ms/step - loss: 94263456.0000 -
mae: 7496.1816 - val_loss: 96434128.0000 - val_mae: 8458.3301
Epoch 37/50
51/51 [==============================] - 0s 3ms/step - loss: 94627200.0000 -
mae: 7529.4351 - val_loss: 81663456.0000 - val_mae: 7837.5181
Epoch 38/50
51/51 [==============================] - 0s 4ms/step - loss: 94659960.0000 -
mae: 7555.7896 - val_loss: 87094728.0000 - val_mae: 8071.9038
Epoch 39/50
51/51 [==============================] - 0s 4ms/step - loss: 93147128.0000 -
mae: 7502.1338 - val_loss: 57796696.0000 - val_mae: 6631.6157
Epoch 40/50
51/51 [==============================] - 0s 3ms/step - loss: 94823112.0000 -
mae: 7494.5386 - val_loss: 77701008.0000 - val_mae: 7662.7500
Epoch 41/50
51/51 [==============================] - 0s 3ms/step - loss: 94284680.0000 -
mae: 7513.1411 - val_loss: 79137392.0000 - val_mae: 7725.4722
Epoch 42/50
51/51 [==============================] - 0s 3ms/step - loss: 94065616.0000 -
mae: 7512.7798 - val_loss: 81705912.0000 - val_mae: 7839.2588
Epoch 43/50
51/51 [==============================] - 0s 3ms/step - loss: 93822424.0000 -
mae: 7494.3262 - val_loss: 87814824.0000 - val_mae: 8101.8086
Epoch 44/50
51/51 [==============================] - 0s 3ms/step - loss: 95960000.0000 -
mae: 7606.0503 - val_loss: 71490080.0000 - val_mae: 7369.9653
Epoch 45/50
51/51 [==============================] - 0s 4ms/step - loss: 92793560.0000 -
mae: 7447.1055 - val_loss: 110016600.0000 - val_mae: 8997.7949
Epoch 46/50
51/51 [==============================] - 0s 3ms/step - loss: 92697632.0000 -
mae: 7511.8218 - val_loss: 69807136.0000 - val_mae: 7289.2363
Epoch 47/50
51/51 [==============================] - 0s 3ms/step - loss: 93855920.0000 -
mae: 7498.9507 - val_loss: 66359440.0000 - val_mae: 7110.1152
Epoch 48/50
51/51 [==============================] - 0s 3ms/step - loss: 94443632.0000 -
mae: 7522.4863 - val_loss: 101227056.0000 - val_mae: 8653.2285
```

```
Epoch 49/50
51/51 [==============================] - 0s 4ms/step - loss: 93561080.0000 -
mae: 7502.9268 - val_loss: 73383320.0000 - val_mae: 7464.5776
Epoch 50/50
51/51 [==============================] - 0s 7ms/step - loss: 94066744.0000 -
mae: 7488.1357 - val_loss: 109115504.0000 - val_mae: 8962.8555
Epoch 1/50
51/51 [==============================] - 0s 6ms/step - loss: 140441.7656 - mae:
140441.7656 - val_loss: 94271.8203 - val_mae: 94271.8203
Epoch 2/50
51/51 [==============================] - 0s 4ms/step - loss: 209494.7188 - mae:
209494.7188 - val_loss: 87118.1719 - val_mae: 87118.1719
Epoch 3/50
51/51 [==============================] - 0s 6ms/step - loss: 144715.9219 - mae:
144715.9219 - val_loss: 96432.9219 - val_mae: 96432.9219
Epoch 4/50
51/51 [==============================] - 0s 4ms/step - loss: 130177.4844 - mae:
130177.4844 - val_loss: 113416.6953 - val_mae: 113416.6953
Epoch 5/50
51/51 [==============================] - 0s 5ms/step - loss: 194514.0938 - mae:
194514.0938 - val_loss: 321367.8125 - val_mae: 321367.8125
Epoch 6/50
51/51 [==============================] - 0s 5ms/step - loss: 202876.4375 - mae:
202876.4375 - val_loss: 225599.1875 - val_mae: 225599.1875
Epoch 7/50
51/51 [==============================] - 0s 5ms/step - loss: 143376.2344 - mae:
143376.2344 - val_loss: 524280.7812 - val_mae: 524280.7812
Epoch 8/50
51/51 [==============================] - 0s 3ms/step - loss: 199369.6406 - mae:
199369.6406 - val_loss: 281388.2500 - val_mae: 281388.2500
Epoch 9/50
51/51 [==============================] - 0s 5ms/step - loss: 173039.3281 - mae:
173039.3281 - val_loss: 294493.7188 - val_mae: 294493.7188
Epoch 10/50
51/51 [==============================] - 0s 4ms/step - loss: 173263.3438 - mae:
173263.3438 - val_loss: 32996.3203 - val_mae: 32996.3203
Epoch 11/50
51/51 [==============================] - 0s 2ms/step - loss: 190222.2031 - mae:
190222.2031 - val_loss: 384350.9375 - val_mae: 384350.9375
Epoch 12/50
51/51 [==============================] - 0s 3ms/step - loss: 135321.0781 - mae:
135321.0781 - val_loss: 41972.5312 - val_mae: 41972.5312
Epoch 13/50
51/51 [==============================] - 0s 3ms/step - loss: 186752.4062 - mae:
186752.4062 - val_loss: 51245.6016 - val_mae: 51245.6016
Epoch 14/50
51/51 [==============================] - 0s 2ms/step - loss: 134597.1250 - mae:
134597.1250 - val_loss: 354943.9688 - val_mae: 354943.9688
```

```
Epoch 15/50
51/51 [==============================] - 0s 2ms/step - loss: 143046.6406 - mae:
143046.6406 - val_loss: 177004.5312 - val_mae: 177004.5312
Epoch 16/50
51/51 [==============================] - 0s 2ms/step - loss: 162104.2344 - mae:
162104.2344 - val_loss: 166945.9844 - val_mae: 166945.9844
Epoch 17/50
51/51 [==============================] - 0s 2ms/step - loss: 229526.4219 - mae:
229526.4219 - val_loss: 394871.1562 - val_mae: 394871.1562
Epoch 18/50
51/51 [==============================] - 0s 2ms/step - loss: 196680.7344 - mae:
196680.7344 - val_loss: 56097.4336 - val_mae: 56097.4336
Epoch 19/50
51/51 [==============================] - 0s 2ms/step - loss: 208819.5625 - mae:
208819.5625 - val_loss: 186347.9062 - val_mae: 186347.9062
Epoch 20/50
51/51 [==============================] - 0s 3ms/step - loss: 135489.7969 - mae:
135489.7969 - val_loss: 212444.1562 - val_mae: 212444.1562
Epoch 21/50
51/51 [==============================] - 0s 3ms/step - loss: 140737.0625 - mae:
140737.0625 - val_loss: 13488.6133 - val_mae: 13488.6133
Epoch 22/50
51/51 [==============================] - 0s 2ms/step - loss: 201536.9688 - mae:
201536.9688 - val_loss: 284340.3750 - val_mae: 284340.3750
Epoch 23/50
51/51 [==============================] - 0s 2ms/step - loss: 220917.5938 - mae:
220917.5938 - val_loss: 389307.4375 - val_mae: 389307.4375
Epoch 24/50
51/51 [==============================] - 0s 2ms/step - loss: 200181.8750 - mae:
200181.8750 - val_loss: 292244.7812 - val_mae: 292244.7812
Epoch 25/50
51/51 [==============================] - 0s 2ms/step - loss: 132121.7188 - mae:
132121.7188 - val_loss: 357816.0312 - val_mae: 357816.0312
Epoch 26/50
51/51 [==============================] - 0s 2ms/step - loss: 195879.3906 - mae:
195879.3906 - val_loss: 9613.9160 - val_mae: 9613.9160
Epoch 27/50
51/51 [==============================] - 0s 3ms/step - loss: 182089.9062 - mae:
182089.9062 - val_loss: 83550.2578 - val_mae: 83550.2578
Epoch 28/50
51/51 [==============================] - 0s 3ms/step - loss: 199196.1562 - mae:
199196.1562 - val_loss: 97419.2031 - val_mae: 97419.2031
Epoch 29/50
51/51 [==============================] - 0s 2ms/step - loss: 190698.3281 - mae:
190698.3281 - val_loss: 243743.5000 - val_mae: 243743.5000
Epoch 30/50
51/51 [==============================] - 0s 2ms/step - loss: 343320.1875 - mae:
343320.1875 - val_loss: 537481.4375 - val_mae: 537481.4375
```

```
Epoch 31/50
51/51 [==============================] - 0s 2ms/step - loss: 191516.7500 - mae:
191516.7500 - val_loss: 377748.0312 - val_mae: 377748.0312
Epoch 32/50
51/51 [==============================] - 0s 3ms/step - loss: 267507.9375 - mae:
267507.9375 - val_loss: 241165.2500 - val_mae: 241165.2500
Epoch 33/50
51/51 [==============================] - 0s 3ms/step - loss: 102087.8984 - mae:
102087.8984 - val_loss: 97167.5391 - val_mae: 97167.5391
Epoch 34/50
51/51 [==============================] - 0s 4ms/step - loss: 186050.9219 - mae:
186050.9219 - val_loss: 391704.6250 - val_mae: 391704.6250
Epoch 35/50
51/51 [==============================] - 0s 3ms/step - loss: 215316.7969 - mae:
215316.7969 - val_loss: 392043.4688 - val_mae: 392043.4688
Epoch 36/50
51/51 [==============================] - 0s 2ms/step - loss: 268664.2500 - mae:
268664.2500 - val_loss: 263648.1875 - val_mae: 263648.1875
Epoch 37/50
51/51 [==============================] - 0s 2ms/step - loss: 231842.3281 - mae:
231842.3281 - val_loss: 695151.6250 - val_mae: 695151.6250
Epoch 38/50
51/51 [==============================] - 0s 3ms/step - loss: 183353.7031 - mae:
183353.7031 - val_loss: 84560.9922 - val_mae: 84560.9922
Epoch 39/50
51/51 [==============================] - 0s 2ms/step - loss: 103593.9688 - mae:
103593.9688 - val_loss: 19895.6172 - val_mae: 19895.6172
Epoch 40/50
51/51 [==============================] - 0s 2ms/step - loss: 192437.4531 - mae:
192437.4531 - val_loss: 129819.7188 - val_mae: 129819.7188
Epoch 41/50
51/51 [==============================] - 0s 3ms/step - loss: 95572.5156 - mae:
95572.5156 - val_loss: 88180.7656 - val_mae: 88180.7656
Epoch 42/50
51/51 [==============================] - 0s 3ms/step - loss: 91953.9219 - mae:
91953.9219 - val_loss: 112222.1953 - val_mae: 112222.1953
Epoch 43/50
51/51 [==============================] - 0s 2ms/step - loss: 154113.1875 - mae:
154113.1875 - val_loss: 418362.2812 - val_mae: 418362.2812
Epoch 44/50
51/51 [==============================] - 0s 3ms/step - loss: 206707.1094 - mae:
206707.1094 - val_loss: 121739.7109 - val_mae: 121739.7109
Epoch 45/50
51/51 [==============================] - 0s 3ms/step - loss: 223654.3906 - mae:
223654.3906 - val_loss: 396402.1875 - val_mae: 396402.1875
Epoch 46/50
51/51 [==============================] - 0s 3ms/step - loss: 142341.0156 - mae:
142341.0156 - val_loss: 222745.3906 - val_mae: 222745.3906
```

```
Epoch 47/50
51/51 [==============================] - 0s 4ms/step - loss: 114159.9375 - mae:
114159.9375 - val_loss: 187471.4844 - val_mae: 187471.4844
Epoch 48/50
51/51 [==============================] - 0s 3ms/step - loss: 238150.2969 - mae:
238150.2969 - val_loss: 62283.4727 - val_mae: 62283.4727
Epoch 49/50
51/51 [==============================] - 0s 3ms/step - loss: 95084.4531 - mae:
95084.4531 - val_loss: 119946.0469 - val_mae: 119946.0469
Epoch 50/50
51/51 [==============================] - 0s 3ms/step - loss: 146004.0000 - mae:
146004.0000 - val_loss: 318044.4688 - val_mae: 318044.4688
Model Test MAE: 11552.7305
Model Test MAE: 389665.1562
8/8 [==============================] - 0s 3ms/step
8/8 [==============================] - 0s 2ms/step
Ensemble Test MAE: 195892.9969
```

[ ]:
```python
# Plot the results
plt.figure(figsize=(14, 7))
plt.plot(y_test.values, label='Actual Bitcoin Price')
plt.plot(ensemble_predictions, label='Ensemble Prediction', linestyle='--')
plt.xlabel('Days')
plt.ylabel('Price')
plt.title('Ensemble Prediction vs Actual Bitcoin Price')
plt.legend()
plt.show()
```