CQF Exam 3

Viranca Balsingh

In partial fulfilment of the Certificate in Quantitative Finance
As offered by Fitch Learning under the supervision of Dr. Paul Wilmott

October 31, 2024

1 Question 1: What is the cost function of Logistic Regression?

Logistic regression is a statistical technique used to predict the probability of a binary outcome based on independent variables. More specifically, the target outcome is 0 or 1. Setting it apart from a linear regression.

Sigmoid Function

Logistic regression uses the Sigmoid function to calculate the probability $P(y = 1X)$:

$$\hat{y} = \sigma(\theta^T X) = \frac{1}{1 + e^{-\theta^T X}} \tag{1.1}$$

where θ represents the parameters of the model, and σ is the sigmoid function, which outputs values between 0 and 1, as shown in Figure 1.1.

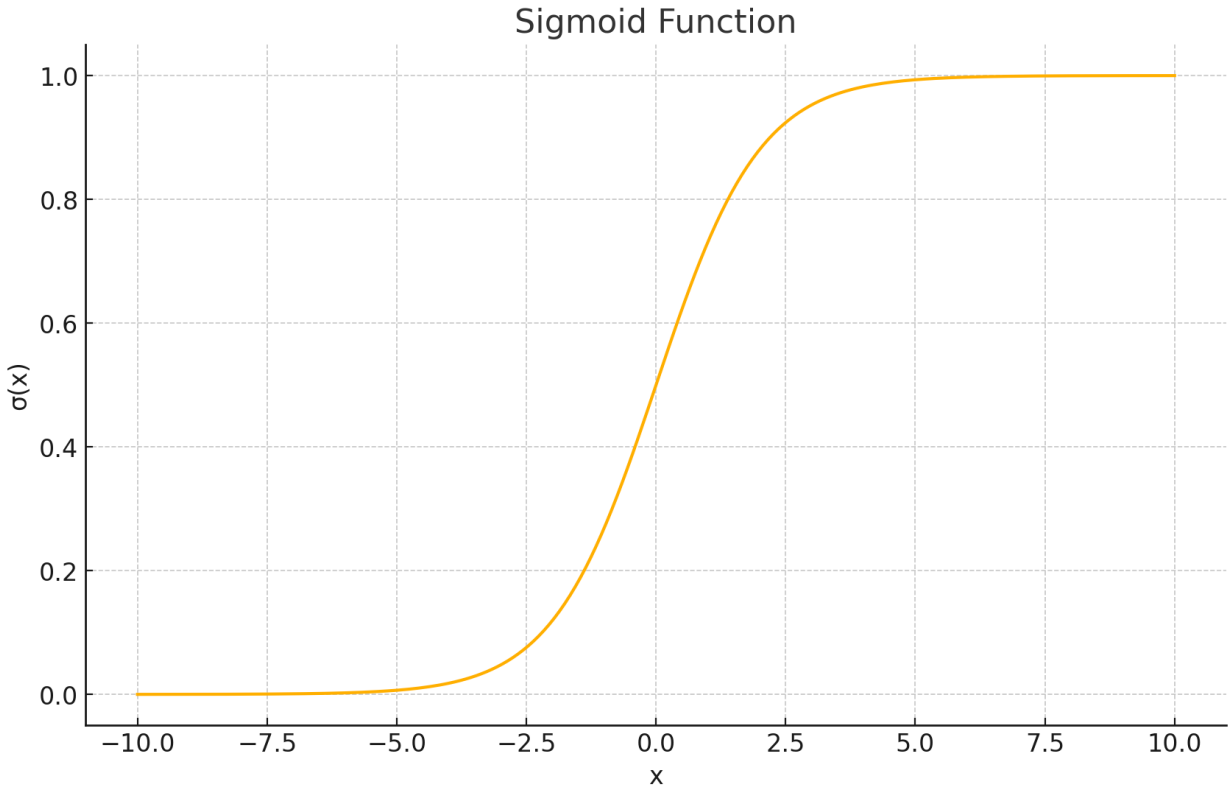


Figure 1.1: Sigmoid Function

Likelihood Function

For a single training example, the probability $P(y|X)$ can be expressed as:

$$P(y|X) = \hat{y}^y \cdot (1 - \hat{y})^{1-y}$$

This expression comes from the fact that:

- If $y = 1$, then $P(y = 1|X) = \hat{y}$.
- If $y = 0$, then $P(y = 0|X) = 1 - \hat{y}$.

This means that for any binary label y, we can use this form to represent the likelihood of the observed label given the model parameters.

Constructing the Log-Likelihood Function

For a dataset with m independent samples $(X^{(i)}, y^{(i)})$, we can write the likelihood function $L(\theta)$ as the product of probabilities for each example:

$$L(\theta) = \prod_{i=1}^{m} P(y^{(i)}|X^{(i)}; \theta) = \prod_{i=1}^{m} \left(\hat{y}^{(i)}\right)^{y^{(i)}} \cdot \left(1 - \hat{y}^{(i)}\right)^{1-y^{(i)}}$$

To simplify, we take the natural logarithm of the likelihood, which gives us the **log-likelihood**:

$$\log L(\theta) = \sum_{i=1}^{m} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\right]$$

Cost Function (Negative Log-Likelihood)

In Logistic Regression, we want to find parameters θ that maximize the log-likelihood, which is equivalent to minimizing the **negative log-likelihood**:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})\right]$$

This expression $J(\theta)$ is the **loss function** or **cost function** for Logistic Regression.

Properties of the Cost Function

Convexity: The cost function is convex, which guarantees that there is a single global minimum. This makes it computationally efficient to find the optimal parameters.
Interpretability: Since the cost function is based on probabilities, it provides an intuitive measure of performance, rewarding accurate predictions and penalizing inaccurate ones.

2 Question 2: What are voting classifiers in ensemble learning?

The Voting Classifier is an ensemble machine learning technique that combines multiple classification models to make predictions, aiming to enhance accuracy by leveraging the strengths of different models. In this approach, individual model predictions are aggregated based on a voting mechanism, either hard or soft.

1. Hard Voting

- Each model makes a discrete class prediction.
- The final class prediction is determined by majority voting across all classifiers. In other words, the class that appears most frequently in the individual predictions is chosen.

Let C_i be the class predicted by the i^{th} classifier. The hard voting prediction $\hat{y}$ is:

$$\hat{y} = \text{mode}(C_1, C_2, \ldots, C_n)$$

where n is the number of classifiers in the ensemble.

2. Soft Voting

- Each model outputs a probability distribution over each class.
- The final class prediction is made by averaging these probabilities across classifiers and selecting the class with the highest probability.

Let $P_{i,j}$ denote the probability of the i^{th} classifier predicting class j. The soft voting prediction $\hat{y}$ is given by:

$$\hat{y} = \arg\max_j \left(\frac{1}{n} \sum_{i=1}^{n} P_{i,j} \right)$$

where j ranges over all possible classes, and n is the number of classifiers.
In practice, Voting Classifiers are powerful when combining diverse models, as they reduce the likelihood of correlated errors and enhance prediction accuracy.

3 Question 3: Follow the 7-steps to model building for your selected ticker

Step 1: Ideation

The goal is to predict short-term (daily) uptrend movements for Copper Futures, as chosen within the commodities remit. Predicting short-term trends for this asset requires careful feature engineering due to the high volatility and low signal-to-noise ratio in short-term price movements. Please refer to the Jupyter Notebook for the full write out of the underlying code. This report only includes the main functions to highlight the flow of the numerical processing.

Step 2: Data Collection

Using the Yahoo Finance API, OHLC (Open, High, Low, Close) data is collected for the last 5 years:

```
import yfinance as yf
from datetime import datetime, timedelta

end_date = datetime.now()
start_date = end_date - timedelta(days=5*365)
ticker = "HG=F"
df = yf.download(ticker, start=start_date, end=end_date)
```

This resulted in 1,256 records of daily price data for Copper Futures, covering the specified 5-year period.

Step 3: Exploratory Data Analysis

Summary statistics and visualizations were created to inspect data distribution:

- The mean and median of daily prices show stability over time, with a moderate level of volatility.
- Log returns are computed to assess the variability of price movements over short periods, revealing a near-normal distribution.

```
df.describe()
```

Step 4: Cleaning Dataset

Data quality checks are performed to identify and handle missing values. The ‘isnull().sum()‘ function revealed a small number of missing values in derived features, which are imputed where appropriate to maintain data consistency:

```
df.isnull().sum()
```

Table 3.1: Descriptive Statistics for Copper Futures (Ticker: HG=F)

Statistic	Price	Open	High	Low	Close
Count	1258	1258	1258	1258	1258
Mean	3.7767	3.8034	3.7494	3.7781	3.7781
Std Dev	0.6576	0.6647	0.6508	0.6576	0.6576
Min	2.1135	2.1255	2.0595	2.1195	2.1195
25%	3.4729	3.5055	3.4501	3.4750	3.4750
Median	3.8490	3.8743	3.8320	3.8523	3.8523
75%	4.2999	4.3305	4.2614	4.2865	4.2865
Max	5.1920	5.1985	5.0575	5.1190	5.1190

Step 5: Feature Engineering

Multiple features were engineered to enhance predictive power:

- **Log Returns:** Defined as:

$$\text{log\_return} = \ln\left(\frac{\text{Close}_t}{\text{Close}_{t-1}}\right)$$

- **Trend Sign:** Classified as 1 if log return exceeds 0.25%, otherwise 0:

```
df['trend_sign'] = (df['log_return'] > 0.0025).astype(int)
```

- **Past Returns:** Lagged returns for the past 10 days were added to capture short-term momentum.
- **Moving Averages:** Calculated Simple Moving Averages (SMA) and Exponential Moving Averages (EMA) for 50, 100, and 200 days.

A heatmap of correlations among OHLC features and engineered features helped identify highly correlated variables, informing later feature selection.

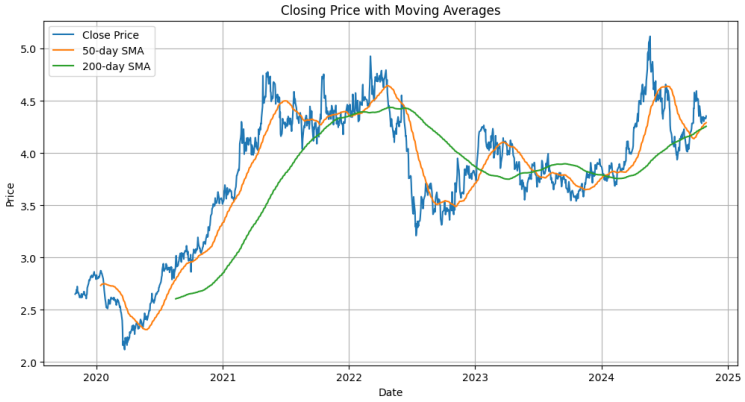


Figure 3.1: Price movement over time with SMA's

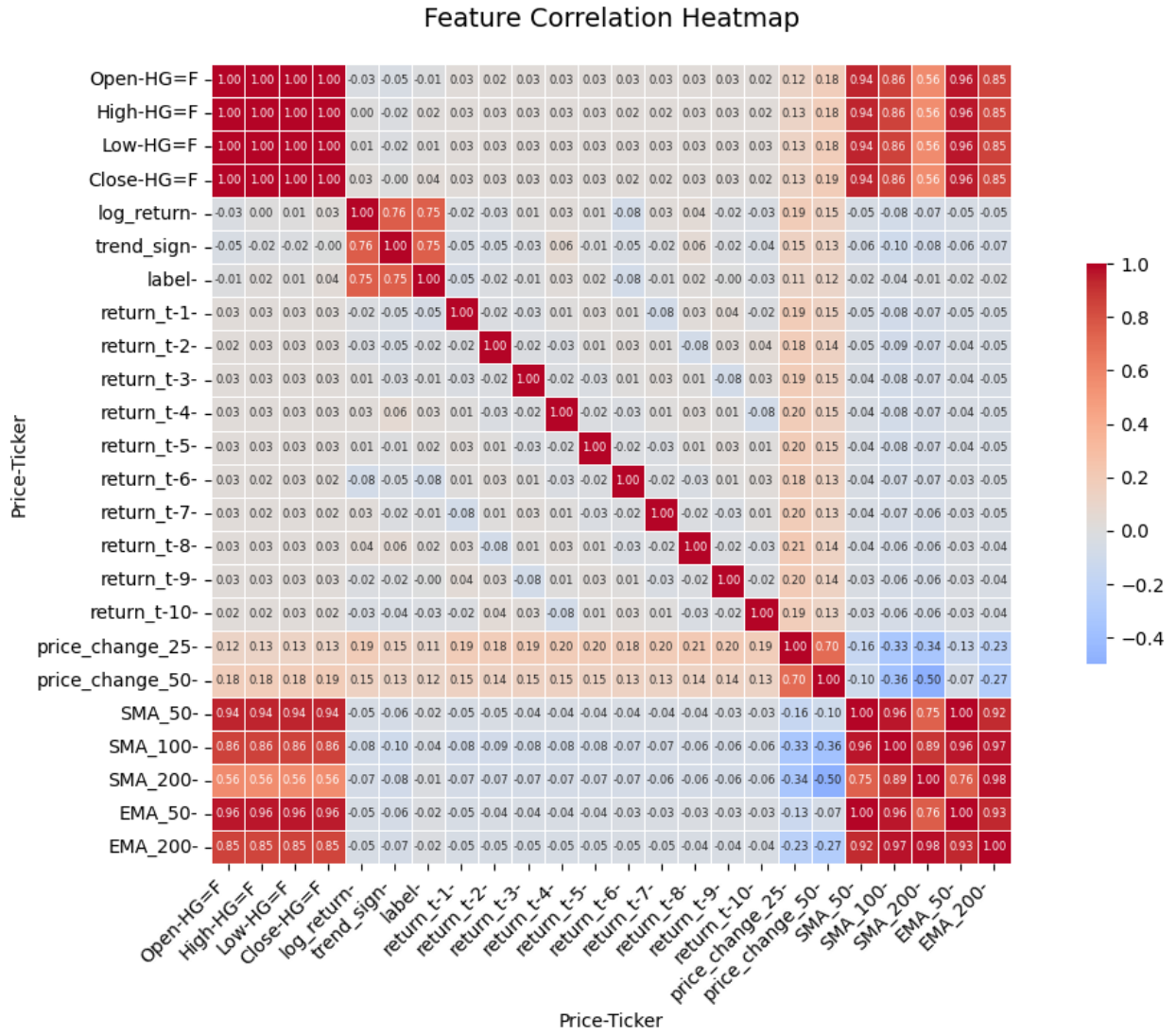
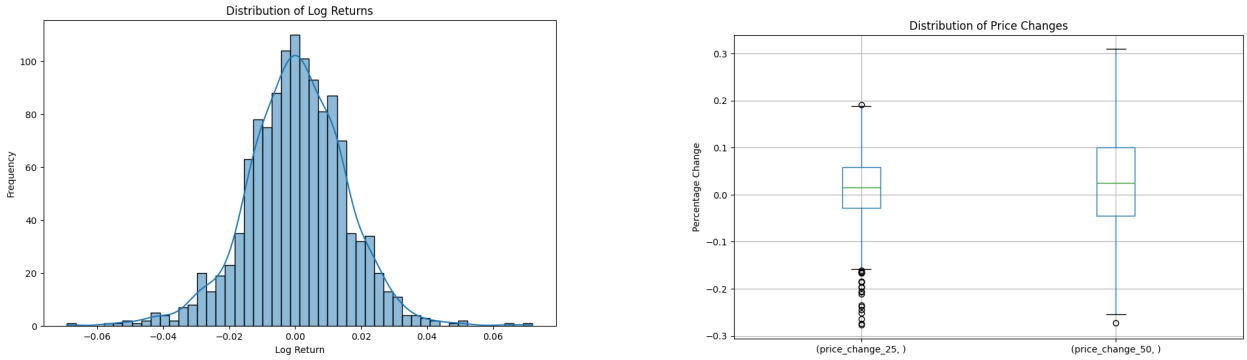


Figure 3.2: Correlation Heatmap of Features



(a) Distribution of Log Returns

(b) Distribution of Price Changes

Figure 3.3: Visualizations of Log Returns and Price Changes

Step 6: Modeling

An SVM model with a radial basis function (RBF) kernel is chosen for this classification task. The feature matrix X and target vector y are created by dropping and selecting the appropriate columns.
After splitting the data, a grid search is performed for hyperparameter tuning:

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 'auto', 0.1, 0.01, 0.001],
    'kernel': ['rbf', 'linear', 'poly'],
    'class_weight': [None, 'balanced']
}
grid_search = GridSearchCV(estimator=SVC(), param_grid=param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)
```

Table 3.2 presents the average performance metrics for various values of parameter C. It includes the mean performance score, standard deviation, and count, showing how increasing values of C affect model stability and performance. Table 3.3 details the performance for different gamma values. It captures the mean, standard deviation, and count, illustrating the impact of gamma on model accuracy, with higher gamma values yielding more stable performance. Table 3.4 compares the performance across different kernel types (linear, polynomial, and RBF). It highlights the mean accuracy, standard deviation, and count, showing that the linear kernel achieved the highest mean performance among the options tested.
The optimal parameters were found to be:[`C`: 10 `Kernel`: linear `Gamma`: scale]

Table 3.2: C Performance Summary

C	Mean	Standard Deviation	Count
0.1	0.770403	0.232837	30
1.0	0.847166	0.190883	30
10.0	0.877598	0.146340	30
100.0	0.892714	0.141866	30

Table 3.3: Gamma Performance Summary

Gamma	Mean	Standard Deviation	Count
0.001	0.695297	0.283314	24
0.01	0.803239	0.228898	24
0.1	0.914633	0.038886	24
auto	0.910820	0.048970	24
scale	0.910862	0.049114	24

Table 3.4: Kernel Performance Summary

Kernel	Mean	Standard Deviation	Count
linear	0.951531	0.040802	40
poly	0.719147	0.233335	40
rbf	0.870233	0.143180	40

Step 7: Evaluation Metrics

The model's performance was evaluated with multiple metrics:

- **ROC Curve and AUC:** Displayed below, illustrating strong discrimination ability.
- **Confusion Matrix:** Showed high counts for true positives and true negatives.
- **Classification Report:** Detailed metrics are included for each class.

The ROC curve shown here evaluates the performance of the classification model used in the report for predicting short-term uptrend movements in Copper Futures. The curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) across different decision thresholds, providing a visual representation of the model's ability to distinguish between positive and negative classes.

The area under the ROC curve (AUC) is 0.77, suggesting that the model has a 77% probability of correctly distinguishing between an uptrend and a non-uptrend event. The ROC curve's upward trajectory, especially above the diagonal random guessing line, reflects that the model performs better than random classification. This AUC value provides a benchmark for evaluating the effectiveness of the model's feature engineering and parameter tuning.

The confusion matrix shows the model accurately predicted 117 downtrends and 133 uptrends, with only two false positives and no false negatives, indicating strong predictive performance.

The Support Vector Machine model, after tuning and evaluation, achieved a high level of accuracy in predicting uptrends for Copper Futures. The modeling process highlighted the importance of detailed feature engineering and hyperparameter optimization, leading to a final model with 98.5% cross-validation accuracy.

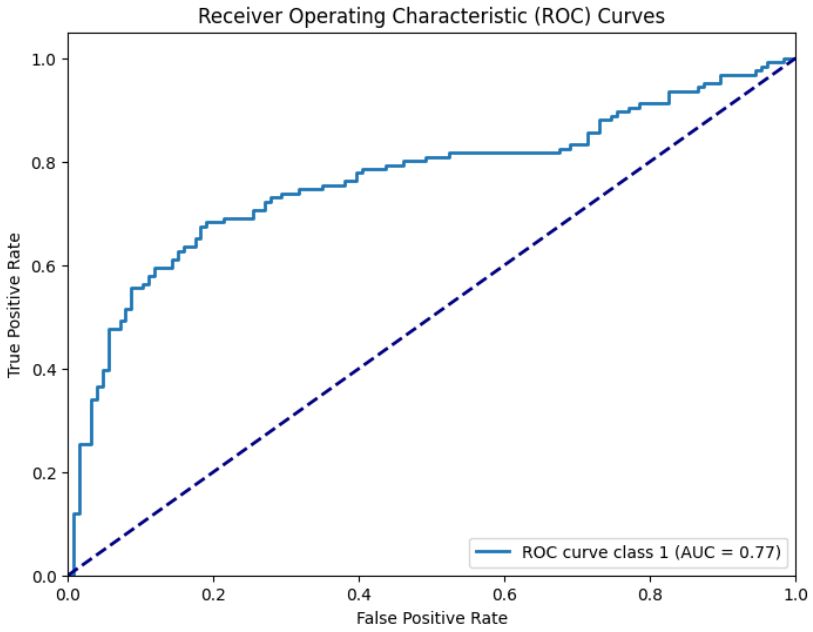


Figure 3.4: ROC Curve for SVM Model

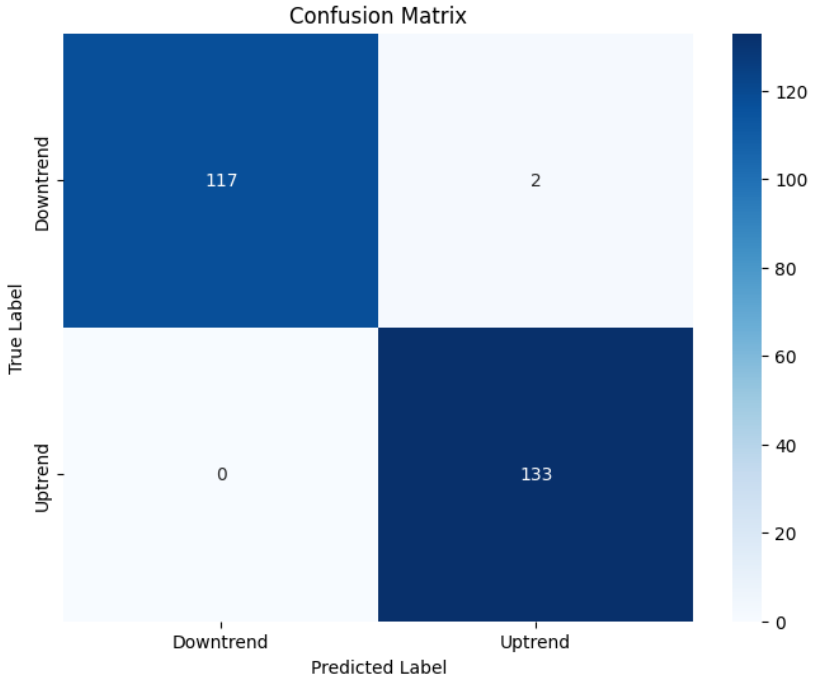


Figure 3.5: Confusion Matrix for SVM Model