Asset Management Student Association
Rotterdam, 2023

Anomaly Detection for Options Pricing Using Autoencoders, Support Vector Machine, & Isolation Forest Methods

Dominika Klepaczko, Filip Borzęcki

Introduction to Options

An option is a derivative financial instrument that represents a right that the option holder possesses, to buy or sell an underlying security (such as stocks). We differentiate two types of options:

1. **Call options**: gives the owner the right, but not the obligation to buy the underlying security in the future, at a price set today.
2. **Put option**: gives the owner the right, but not the obligation to sell the underlying security at a price set today. The seller of a put option is obligated to buy if asked.

Since the value of an option is based on the underlying stock, we can track how change in the latter affects the former (refer to *Figure 1*).

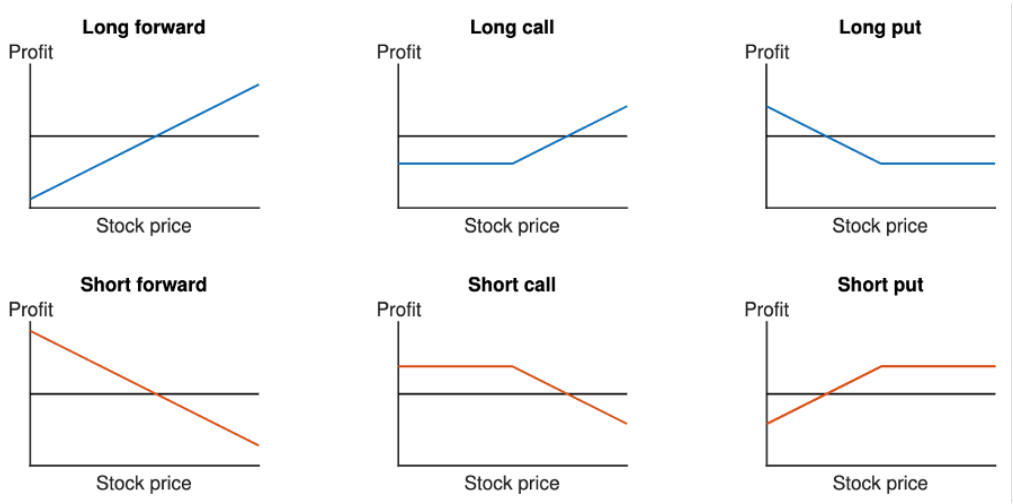


***Figure 1**: Stock price and profit for different types of options.*

Option pricing theory

Models

In option pricing theory, the assessment of option's value relies on several factors. Black and Scholes (1973) identify a model which incorporates five key variables, namely: underlying price, volatility of the underlying asset, strike price, time to maturity and risk-free interest rate. This influential model has significantly shaped the theory of option pricing. Nevertheless, like most models, this model makes a lot of assumptions that are not very realistic. For the model to be accurate, the volatility of an underlying asset and risk-free rate must be constant. Moreover, there cannot be any dividends and the model can be applied to the European options only. Although we recognize the importance of the Black-Scholes variables in the option pricing, we decide to go a step further and also incorporate option Greeks in our model.

Option Greeks

Option Greeks represent the sensitivity of the option price to various factors, including stock price, volatility, interest rate, and time to maturity. Those variables are denoted by letters from the Greek alphabet, including:

Delta: a change in option price when stock price increases by 1$.

Gamma: a change in delta when option price increases by $1.

Vega: change in option price when volatility increases by 1%.

Theta: change in option price when time to maturity decreases by 1 day.

Rho: change in option price when interest rate increases by 1%.

Anomaly Detection

Anomaly detection is a term, which covers a wide range of methods that are used to spot unexpected data points in a dataset. We define an anomaly as an event with features that are significantly different from what can be observed in most of the data and which occurs relatively rarely. Since, as described above, option pricing is a difficult task that relies on using a wide variety of variables, we expect such anomalies to be present in a dataset of options. Spotting those anomalies could be a base for profitable trades or a warning for option holders. Instead of using model-based approaches to these tasks, which are just generalizations, we will use data-based techniques from the world of neural networks and machine learning, hoping to see some valuable results.

Data

Data for both put and call options were retrieved from the *Historical Option Data* website and is being used for research purposes only. The dataset covers relatively many observations of U.S. Equity options with their respective prices, spreads, and greeks. We will augment the data with additional features and perform some data cleaning on the set. Moreover, an expected fraction of anomalies will be set at a level of 4.65%. Which represents which approximates the fraction of data that lies outside of the range of two standard deviations from the mean. This parameter will stay constant in each of the models.

Libraries

```
# Importing libraries.

# Basic data-analysis and plotting libraries.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Data processing libraries.
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

# Importing Keras framework based on TensorFlow for neural networks.
import tensorflow as tf
from tensorflow.keras.models import Model

# Libraries for SVM.
from sklearn.svm import OneClassSVM

# Libraries for Isolation Forest.
from sklearn.ensemble import IsolationForest
```

Data transformation

```
# Reading the .csv dataset from historicaloptiondata.com
df = pd.read_csv('dataset.csv')

# Creating a feature representing time remaining untill expration (in days).
df[['Expiration',' DataDate']] = df[['Expiration',' DataDate']].apply(pd.to_datetime)
df['Till Expiration'] = (df['Expiration'] - df[' DataDate']).dt.days

# Creating features representing spreads.
df["Ask - Bid"] = df["Ask"] - df["Bid"]
df["Price - Strike"] = df["UnderlyingPrice"] - df["Strike"]

# Preparing a dataframe for dummy variables created from the type of options.
df_type = pd.get_dummies(df["Type"])

# Selecting relevant features.
df = df[["Type", "Till Expiration", "UnderlyingPrice", "Strike", "Price - Strike", "Ask - Bid", "Delta", "Gamma", "Theta", "Vega"]]

# Augmenting the main dataframe with the dummy variables.
pd.options.mode.chained_assignment = None
df["Call/Put"] = df_type["call"]
df = df.drop(columns=["Type"])

# Removing missing values.
imputer = SimpleImputer(strategy='mean')
df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

```
# Scaling the data.
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

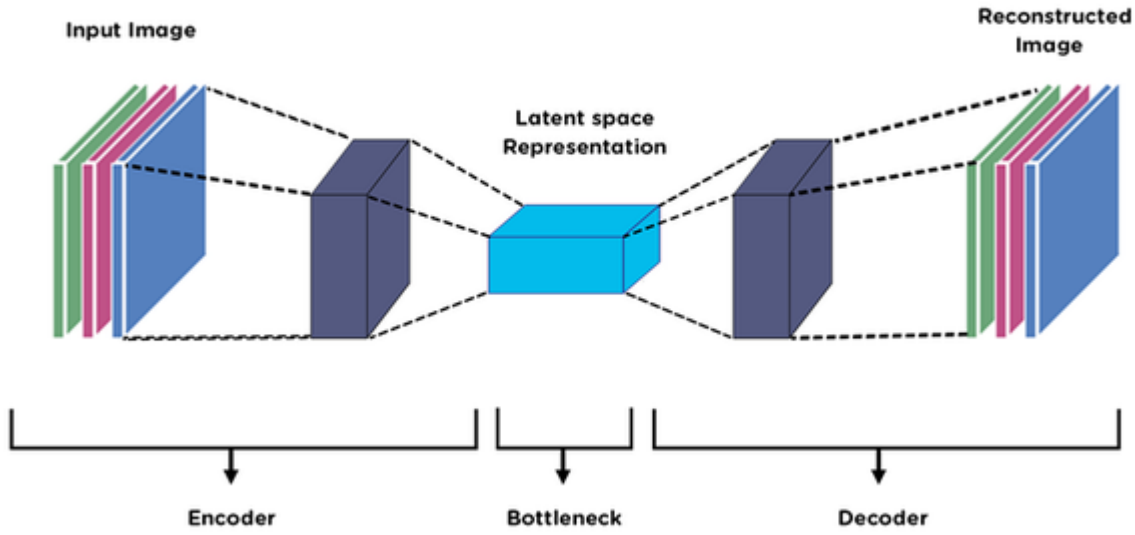
Dataset

```
df.head()
```

	Till Expiration	UnderlyingPrice	Strike	Price - Strike	Ask - Bid	Delta	Gamma	Theta	Vega	Call/Put
0	1.0	69.59	42.5	27.09	4.45	1.0	0.0	-0.9762	0.0	1.0
1	1.0	69.59	42.5	27.09	0.04	0.0	0.0	0.0000	0.0	0.0
2	1.0	69.59	45.0	24.59	4.65	1.0	0.0	-1.0336	0.0	1.0
3	1.0	69.59	45.0	24.59	0.04	0.0	0.0	0.0000	0.0	0.0
4	1.0	69.59	47.5	22.09	4.55	1.0	0.0	-1.0910	0.0	1.0

Autoencoders

Autoencoder is a neural network architecture composed of two functions: encoding function and decoding function - both of them are in the form of an artificial neural network. The main goal of the encoding function is to compress the data to a predetermined size (using an appropriate number of layers). Then, the decoding function tries to recreate the original data from the compressed version based on what it saw in the training. This process is "lossy", which means that accuracy and quality of the recreated data is lost to a certain extent. The loss gets higher when the network encounters data that it is not used to and, thus, "struggles" with proper recreation. We will use that property to determine a threshold for loss, which will indicate which observations are anomalous and which are normal.



Source: Medium

Model Set-up

```
# Creating the autoencoder model.
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(10, activation="relu"))
model.add(tf.keras.layers.Dense(6, activation="relu"))
model.add(tf.keras.layers.Dense(4, activation="relu"))
model.add(tf.keras.layers.Dense(1, activation="relu"))
model.add(tf.keras.layers.Dense(4, activation="relu"))
model.add(tf.keras.layers.Dense(6, activation="relu"))
model.add(tf.keras.layers.Dense(10, activation="relu"))
model.add(tf.keras.layers.Dense(10, activation="sigmoid"))
```

```
# Defining the condition for early stopping to avoid overfitting.
early_stopping = tf.keras.callbacks.EarlyStopping(monitor="loss", patience=2, mode="min")

# Compiling the model with ADAM optizmizer and Mean Absolute Error as the loss funciton.
model.compile(optimizer='adam', loss="mae")

# Training the model in ten epochs each with a batch size of 32 and storing the results in the "history" object.
history = model.fit(df_scaled, df_scaled, epochs=10, batch_size=32, shuffle=True, callbacks=[early_stopping])
```

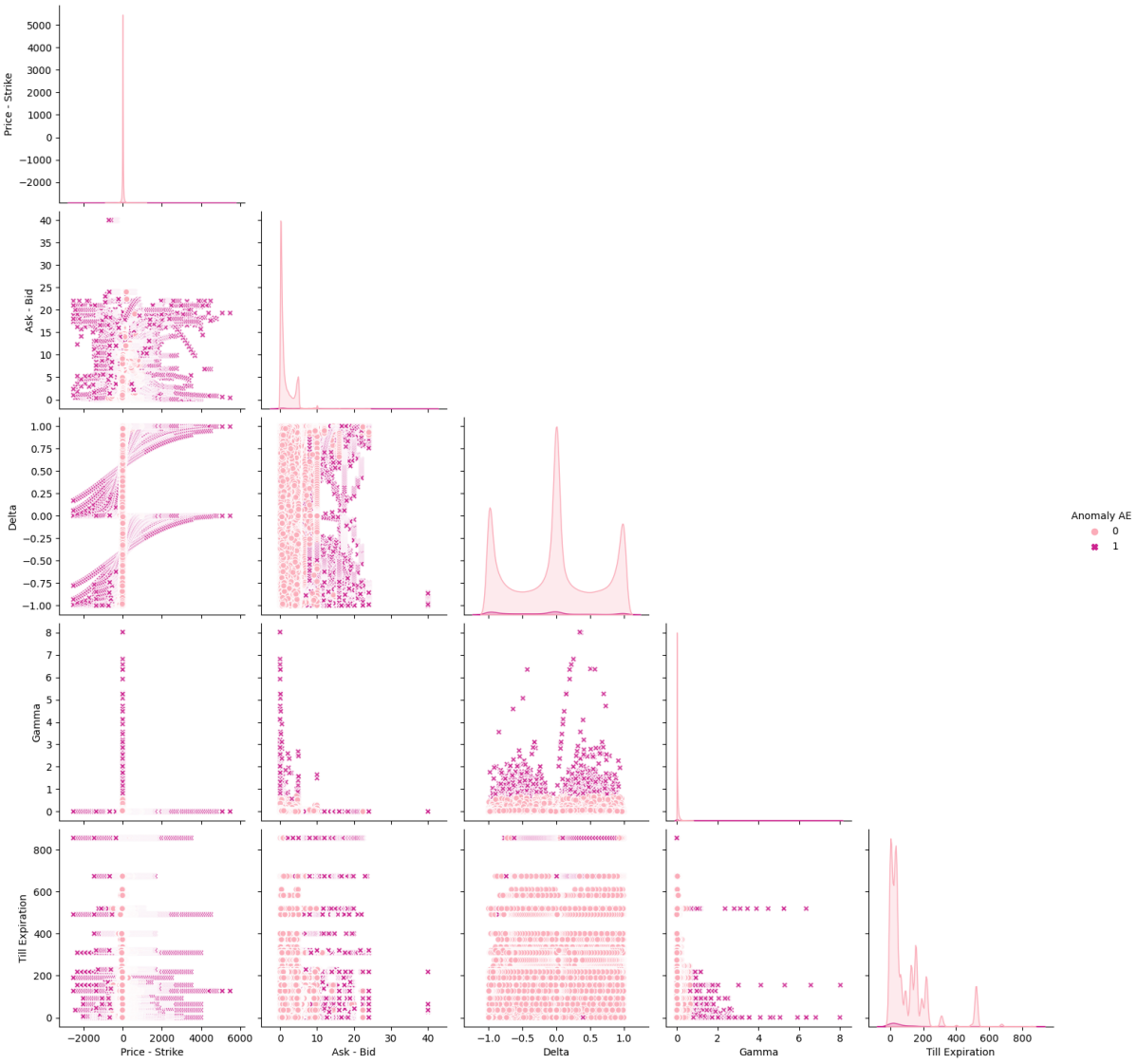
Model Evaluation

```
# Using the training model to reconstruct the previously compressed data.
reconstructed = model.predict(df_scaled)

# Calculating the loss during reconstruction of each of the observations.
train_loss = tf.keras.losses.mae(reconstructed, df_scaled)
```

```
# Creating an array storing loss for each observation.
train_loss_array = np.array(train_loss)

# Setting up an arbitrary threshold of two standard deviations from the average loss.
threshold = np.mean(train_loss) + 2*np.std(train_loss)
anomalies = pd.DataFrame(train_loss_array)

# Appending loss and verdict on anomaly to the main dataframe.
df["Loss"] = anomalies[0]
df["Anomaly AE"] = np.where(df['Loss']>=threshold, 1, 0)

# Calculating the percantage of anomalies detected by the model.
percentage_AE = (np.sum(df["Anomaly AE"])/len(df.index))*100
print(f'The model detects {percentage_AE}% of observations as anomalies')
```

```
The model detects 2.9279381139129557% of observations as anomalies
```

Graphical Representation of the Results

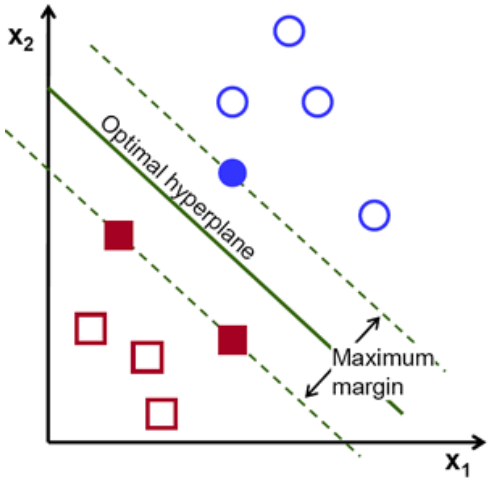


In our sample, we can observed that the selected factors

- **Price - Strike spread**: normal observations are accumulated around zero. More positive or negative values are recognized as outliers.
- **Ask - Bid spread**: normal observations are mostly recognized for range 0-10, more extreme values are most often seen as outliers.
- **Delta**: does not seem to have a significant impact on the outliers. The normal observations are spread across the whole range of Delta values. In the graphs, outliers are mostly determined by the other variable used in scatter with Delta.
- **Gamma**: outliers can be detected for the higher values of Gamma, starting around 0.75.
- **Till Expiration**: Outliers can be seen mostly for very high maturities, around 700 days.

Support Vector Machines

Support Vector Machine (SVM) is a widely employed model in anomaly or outlier detection tasks. The regular version of SVM is a supervised model meaning that it relies on the target labels during the model training. In our project, we are going to use an unsupervised technique called One-Class Support Vector Machine. This model will automatically learn the boundary to differentiate between the normal data points and anomalies (Medium, 2022). It will allow us to detect the anomalies without the need to for labeling the data.



Source: tomaszkacmajor.pl

In Python language this model can be applied using a OneClassSVM function from sklearn.svm library. This function among others takes the argument of kernel and nu. Kernel specifies the type of algorithm and was set to default. Nu represents the lower bound of the fraction of support vectors and bound on the fraction of training errors and was set to 4.65% to match with other models (Scikit Learn, n.d.a).

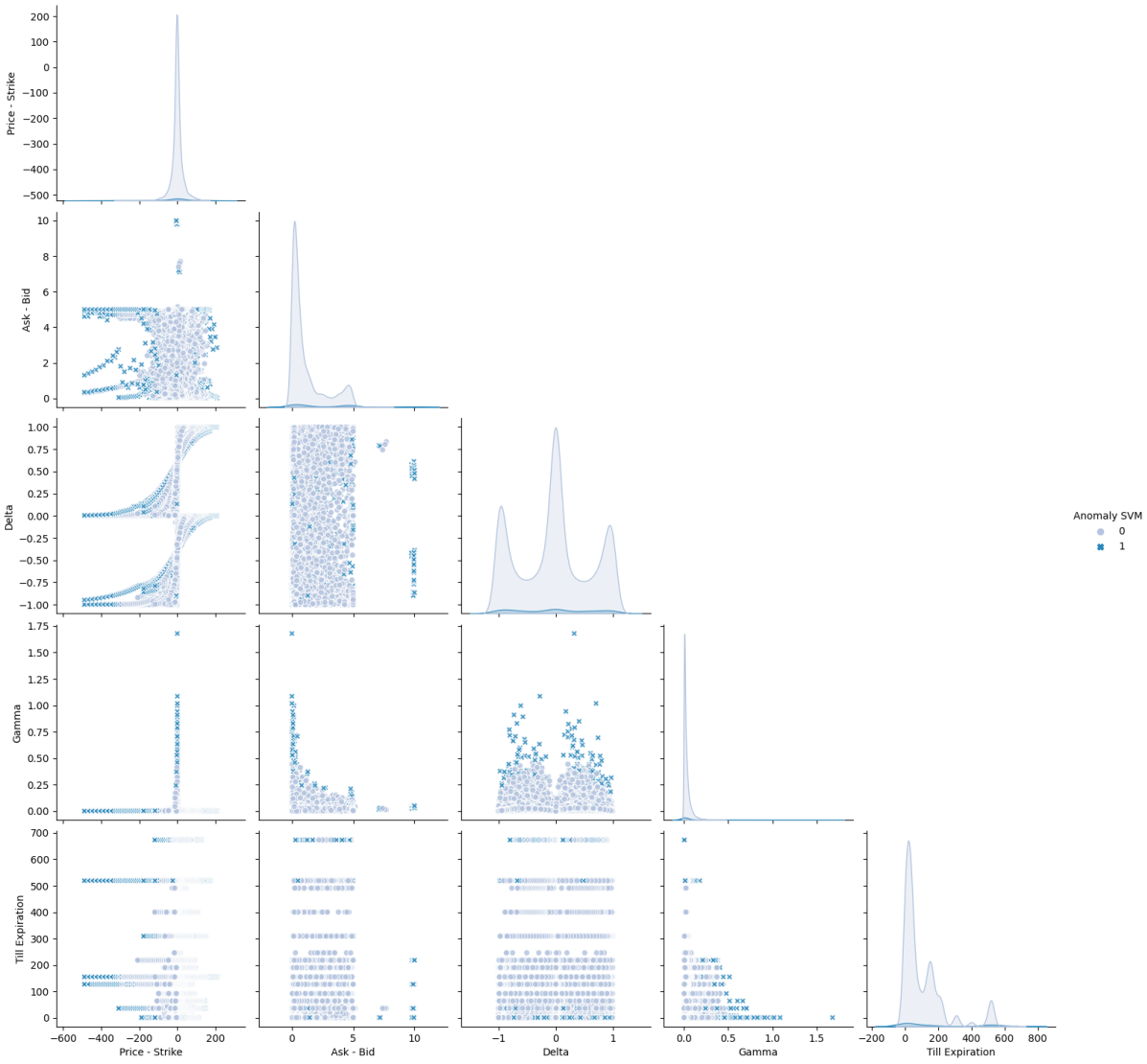
Model Set-up

```
# SVM model set-up.
svm = OneClassSVM(kernel='rbf', nu=0.0465).fit(df_scaled)
```

Model prediction

```
# Prediction using SVM.
prediction_svm = svm.predict(df_scaled)

# Creating a feature representing anomalies detected by SVM.
prediction_svm = [1 if i==-1 else 0 for i in prediction_svm]
df['Anomaly SVM'] = prediction_svm

# Calculating the percantage of anomalies detected by the model.
percentage_SVM = (np.sum(df["Anomaly SVM"])/len(df.index))*100
print(f'The model detects {percentage_SVM}% of observations as anomalies')
```

Graphical Representation of the Results

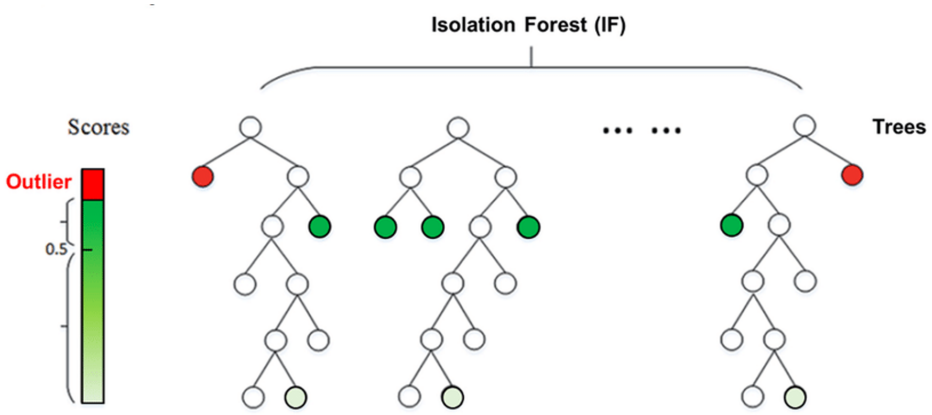


In our sample, we can observed that the selected factors:

- **Price - Strike spread**: normal observations are accumulated around zero. More positive tend to also be normal but negative values usually are recognized as outliers.
- **Ask - Bid spread**: normal observations are mostly recognized for range 0-5 with few exceptions, more extreme values are most often seen as outliers.
- **Delta**: does not seem to have a significant impact on the outliers. The normal observations are spread across the whole range of Delta values. In the graphs, outliers are mostly determined by the other variable used in scatter with Delta.
- **Gamma**: outliers can be detected for the higher values of Gamma, starting around 0.5.
- **Till Expiration**: Outliers can be seen mostly for very high maturities, around 700 days and for maturities around 0.

Isolation Forest

Isolation Forest isolated observations through a random selection process. It randomly selects a feature and then selects a split value within a range of minimum and maximum values of that selected feature. It owes its name to the tree structure created by the recursive partitioning. The number of splits required to isolate a sample represents the path length from the root node to the terminating node. The path length serves as a measure of normality and is crucial in the decision function. In case of anomalies, random partitioning generates much shorter paths. Consequently, if the forest of random trees produces shorter path lengths for a sample, it indicates that those are likely to be anomalies (Scikit Learn, n.d.b).



Source: Isolation Forest Model

To employ this method IsolationForest must be downloaded from the sklearn.ensemble library. This function takes arguments of max_samples, contamination, n_estimators, random_state, max_features. Contamination represents the proportion of outliers in the data set used to define the threshold on the scores of the sample and was set to 4.65% to match with the other models. The rest is presented in the code.

Model Set-up

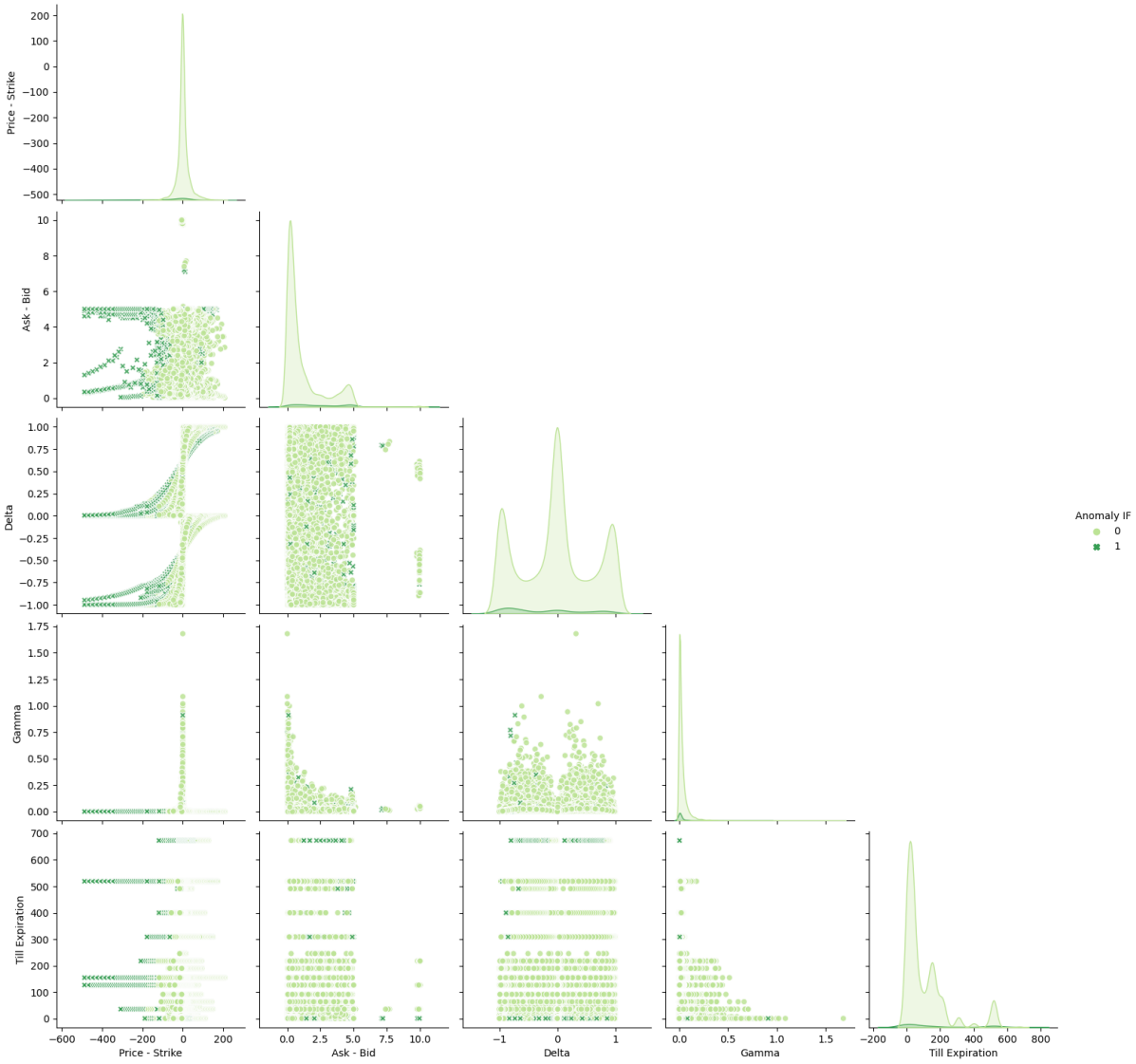
```
# Isolation Forest model set-up.
forest = IsolationForest(max_samples='auto', contamination=.0465, n_estimators=10, random_state=19117, max_features=df_scaled.shape[1])

# Training the model.
forest.fit(df_scaled)
```

Model prediction

```
# Predicting using IF.
prediction_IF = forest.predict(df_scaled)

# Creating a feature representing anomalies detected by IF.
prediction_IF = [1 if i==-1 else 0 for i in prediction_IF]

# Adding this feature to the dataframe.
df['Anomaly IF'] = prediction_IF

# Calculating the percantage of anomalies detected by the model.
percentage_IF = (np.sum(df["Anomaly IF"])/len(df.index))*100
print(f'The model detects {percentage_IF}% of observations as anomalies')
```

```
The model detects 4.645763744866336% of observations as anomalies
```

Graphical Representation of the Results



In our sample, we can observed that the selected factors:

- **Price - Strike spread**: normal observations are accumulated around zero. More positive tend to also be normal but negative values usually are recognized as outliers.
- **Ask - Bid spread**: normal observations are mostly recognized for range 0-5, more extreme values are most often seen as outliers.

- **Delta**: does not seem to have a significant impact on the outliers. The normal observations are spread across the whole range of Delta values. In the graphs, outliers are mostly determined by the other variable used in scatter with Delta.
- **Gamma**: does not seem to have a significant impact on the outliers. Only few outliers can be seen on the graphs and they do not represent any pattern.
- **Till Expiration**: Outliers can be seen mostly for very high maturities, around 700 days and for maturities around 0.

Conclusion

The three different approaches we examined in this paper are based on different mathematical and statistical mechanisms and, as we observed, produce different results. We, however, see many practical use-cases for each of them that could be further examined, such as building a trading strategy based on detected anomalies that could be exploited to achieve returns. In our analysis, we were able to separate anomalous observations from normal ones and represent differences between them visually. Moreover, we spotted certain patterns and relationships that are present for both types of observations. In each model, we could observe a very small impact of delta on the distribution of anomalies. Further, similar patterns could be seen for time to maturity, where the outliers were usually recognized for the maturity of more than 700 days. Nevertheless, the models also exhibited many differences. Those were not that significant for the Isolation Forest method and SVM. Between the two models, the biggest difference was seen in the impact of gamma. While, in Isolation Forest, it had little to no impact on distribution of outliers, in SVM, observations that had gamma higher than 0.5 were mostly recognized as outliers.

Lastly, we recognize that the analysis is not yet complete and we see a lot of room for further examination of optimal choice of hyperparameters and comparisons of the machine learning-based results with those of classical models.

Bibliography

Black, F., Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. T*he Journal of Political Economy. 81*(3). 637-654.

Datrics AI. (n.d.). Isolation Forest Model. Retrieved on June 10, 2023, from https://wiki.datrics.ai/isolation-forest-model.

Github. (2019, February 3). Unsupervised Learning for anomaly detection in option pricing. https://github.com/borisbanushev/anomaliesinoptions.

Kacmajor, T. (2016, April 17). Support Vector Machine. tomaszkacmajor.pl. https://tomaszkacmajor.pl/index.php/2016/04/17/support-vector-machine/.

Medium. (2019, March 12). Basics of Autoencoders. https://medium.com/@birla.deepak26/autoencoders-76bb49ae6a8f.

Medium. (2022, March 25). One-Class SVM for Anomaly Detection. https://medium.com/grabngoinfo/one-class-svm-for-anomaly-detection-6c97fdd6d8af.

Scikit Learn. (n.d.a). sklearn.svm.OneClassSVM. Retrieved on June 10, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html.

Scikit Learn. (n.d.b). sklearn.ensemble.IsolationForest. Retrieved on June 10, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html.

Singh, G. (2021, June 15). A Gentle Introduction to Autoencoders for Data Science Enthusiast. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/autoencoders-a-gentle-introduction/.

Appendix A Code for graphical representation of the results

```
# Choosing features to plot.
sns_options_vars_svm = ['Price - Strike', 'Ask - Bid', 'Delta', 'Gamma','Till Expiration']

# Plotting graphs.
options_svm = sns.pairplot(df, vars=sns_options_vars_svm, hue='Anomaly', markers=["o", "X"], diag_kind='kde',
                           plot_kws=dict(edgecolor='white', linewidth=.85, alpha=.85), diag_kws=dict(shade=True),
                           height=3, palette="PuBu")

# Hidding upper part of plots.
for i, j in zip(*np.triu_indices_from(options.axes, 1)):
    options_svm.axes[i, j].set_visible(False)

# Setting labels based on column names.
for i, x in enumerate(sns_options_vars_svm):
    for j, y in enumerate(sns_options_vars_svm):
        options_svm.axes[j, i].xaxis.set_label_text(x)
        options_svm.axes[j, i].yaxis.set_label_text(y)

plt.show()
```