

Final Machine Learning Project

Classifying Good and Bad Credit Risks

Ananya Shah, Ada Wong
December 2024

Introduction

How do banks predict the risk of lending to a person? Based on a set of features, how well can they predict whether a person's credit risk is "good" or "bad"?

To tackle this binary classification problem, we use a dataset taken from OpenML named "credit-g". It contains data that dates back to 1994 Germany, where 1000 people are labelled as "good" or "bad" for credit risks, accompanied by a set of 20 attributes listed here:

1. Status of existing checking account, in Deutsche Mark.
2. Duration in months
3. Credit history (credits taken, paid back duly, delays, critical accounts)
4. Purpose of the credit (car, television,...)
5. Credit amount
6. Status of savings account/bonds, in Deutsche Mark.
7. Present employment, in number of years.
8. Installment rate in percentage of disposable income
9. Personal status (married, single,...) and sex
10. Other debtors / guarantors
11. Present residence since X years
12. Property (e.g. real estate)
13. Age in years
14. Other installment plans (banks, stores)
15. Housing (rent, own,...)
16. Number of existing credits at this bank
17. Job
18. Number of people being liable to provide maintenance for
19. Telephone (yes,no)
20. Foreign worker (yes,no)

The aim of the project is to train several machine-learning models on this dataset, evaluating them on data set aside for testing. We will compare and contrast each model, apply techniques of feature transformation and hyperparameter-tuning and study their effects. We will also conduct unsupervised learning on the dataset to shed light on patterns and structures within the data. Altogether, we hope to find a well-performing model with good test accuracies.

To help us build machine-learning models and process data, this project relies on *scikit-learn* and *Pandas*' tools. Matrices are visualized using the Python data visualization library *seaborn*, while graphs are plotted with Matplotlib.

Unsupervised Analysis

Firstly, dimensionality reduction was performed PCA, with an expectation for it to capture 95% of the dataset's variance:

```
# apply dimensionality reduction using PCA

pca = PCA(n_components=0.95)
transformed_data = pca.fit_transform(scaled_data)

pca_df = pd.DataFrame(data=transformed_data, columns=[f"PC{i+1}" for i in
range(transformed_data.shape[1])])
```

The shape of the transformed data was (1000, 10), indicating that we need a minimum of 10 components to capture that level of variance. The data was then stored in a DataFrame, after which KMeans Clustering was applied.

After applying KMeans Clustering, we explored the patterns and similarities in the data. We grouped the data through clustering, and observed how it worked alongside the original dataframe. The mean of each principal component was computed, as a metric for further analysis. Lastly, the numerical features data was extracted, separated by clusters, and was used to output the mean for each dataset:

```
cluster    PC1    PC2    PC3    PC4    PC5    PC6    PC7  \
0      2.002902 -0.591127 -0.101216 -0.006041  0.079356  0.100010  0.060597
1     -0.760495 -0.736339  0.104628 -0.152483 -0.027791 -0.088228 -0.052392
2     -0.147303  1.370872 -0.035070  0.262775 -0.007263  0.018212  0.037962

cluster    PC8    PC9    PC10
0     -0.231866 -0.117050 -0.047522
1      0.021743  0.041590  0.057635
2      0.104215  0.009878 -0.052868
checking_status  duration  credit_amount  savings_status  \
cluster
0      1.056410   38.343590   7551.410256      0.974359
1      1.034115   16.304904   2219.654584      1.247335
2      0.922619   17.199405   2255.104167      1.235119

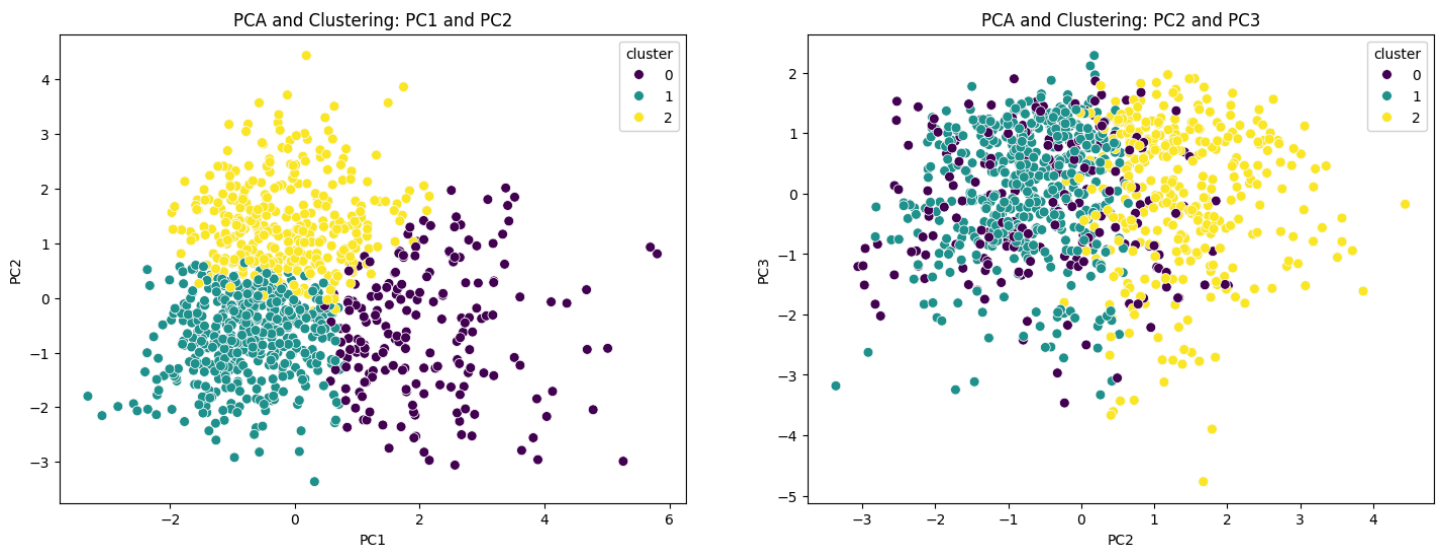
cluster  employment  installment  commitment  residence_since  age  \
0      2.292308      2.707692      2.861538   35.241026
1      1.812367      2.876333      2.356077   29.349680
2      3.235119      3.261905      3.517857   44.372024

cluster  existing_credits  job  num_dependents  class
0      1.456410   2.276923      1.158974   0.548718
1      1.228145   1.793177      1.089552   0.703625
2      1.627976   1.842262      1.244048   0.782738
```

From the table, we could understand the key characteristics in each cluster, in addition to which features most significantly help to differentiate them.

Observing the first part of the table, PC1 and PC2 had distinctive means across each cluster, indicating that it effectively separates the various groups in a dataset. PC3, however, had cluster means that were less distinct, meaning that it could provide some level of separation, but not as much as the first two components. As for the remaining components, the means are not that different across each cluster, and are close to 0. Because of that, they would be ineffective in separating various groups in the dataset.

As a result, we went ahead and explored the structure of PC1, PC2, and PC3, and obtained the following graphs:



As per the table, the separation of PC1 and PC2 was far clearer, in comparison to the overlap between PC2 and PC3. Therefore, PC1 and PC2 capture more meaningful differences and separations, while PC2 and PC3 share more areas of similarity.

From the second half of the table, we gained a deeper understanding into key characteristics of each cluster.

Cluster 0 generally had high credit amounts and duration, indicating that individuals in this group take higher loans and therefore would take longer to repay them. Demographically, they were middle-aged, with an average age of 35.24 who had moderate employment experience of around 2.29 years. Financially, they had a low savings status of 0.97, which indicates limited funds. As a result, they were at slight credit risk, shown through a rating of 0.548.

Cluster 1 had the lowest credit amounts of any cluster, along with the shortest duration. As a result, they generally consisted of individuals who took smaller loans, and were quicker to repay them. In comparison to Cluster 0, they were younger, with an average age of 29.35 years - the youngest of all clusters, who had the smallest amount of employment experience (1.81 years). Financially, they had the highest savings status of all groups, and a stronger class rating of 0.70. As a result, despite their young age, they have moderate financial stability.

Cluster 2 had a moderate credit amount in comparison to clusters 0 and 1, along with a moderate duration. Demographically, they were the oldest with a mean age of 44.37 years, and the highest employment experience of 3.24 years. This led to a high savings status of 1.24 and a class rating of 0.78, suggesting strong financial health and low credit risk.

From the findings obtained in each clustering, it was credit amount, employment, age, duration, and savings_status that played a key role in grouping the data.

After exploring the dataset through clustering and dimensionality, we visualized the training set in several ways: looking at the individual features, the relationship between each feature to the target variable, and relationships between the features.

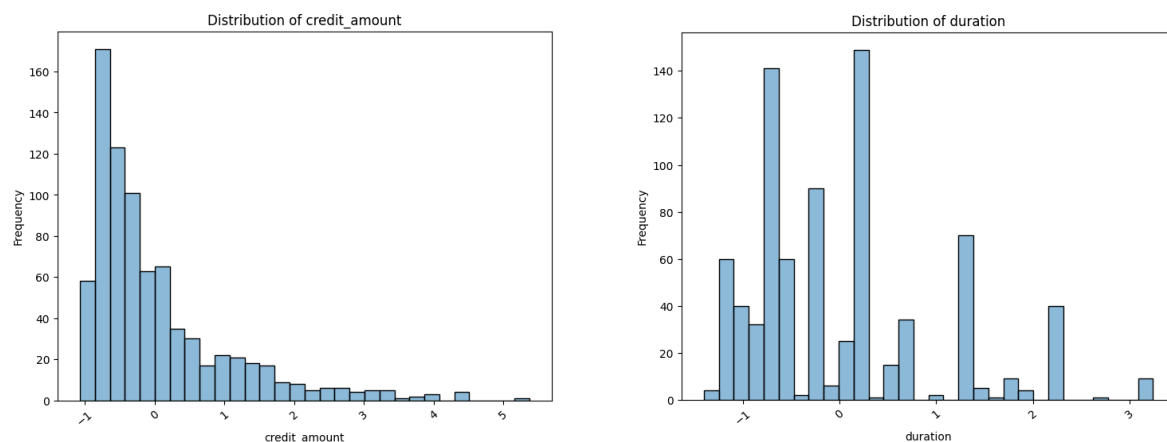
For each feature, we looked at its frequency distribution:

```
# Visualizing individual features

for feature in X_train.columns:
    plt.figure(figsize=(8,6))
    sns.histplot(X_train[feature],kde=False,bins=30, alpha=0.5)
    plt.xlabel(feature)
    plt.xticks(rotation=40)
    plt.ylabel("Frequency")
    plt.title(f"Distribution of {feature}")
    plt.show()
```

In order to draw key findings, we explored each distribution to observe any outlier values, where the maximum lies, and if there was any portion of the data that was skewed. Additionally, as per the insights we drew from the PCA and KMeans Clustering analysis, we looked at distributions for credit amount, age, employment, savings status, and duration for any possible further insights.

While we generated distributions for all features, these were a few that shaped our findings:



Key Findings:

- The credit amount is skewed right meaning that the majority of individuals take smaller loans, but there are a few who take much higher loans.
- The duration distribution has a generally higher frequency for smaller values, peaking at values like 0 and -1. However, there are a few peaks for longer durations, indicating that short-term loans are more common, but there is a slight demand for longer loans.

To analyze the relationship between each feature to the target variable, we utilized a box plot for numerical features, and a count plot for categorical features:

```
numerical_features = X_train.select_dtypes(include=['number'])

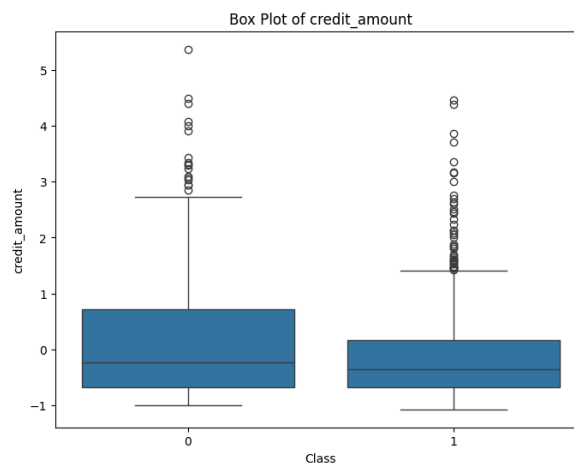
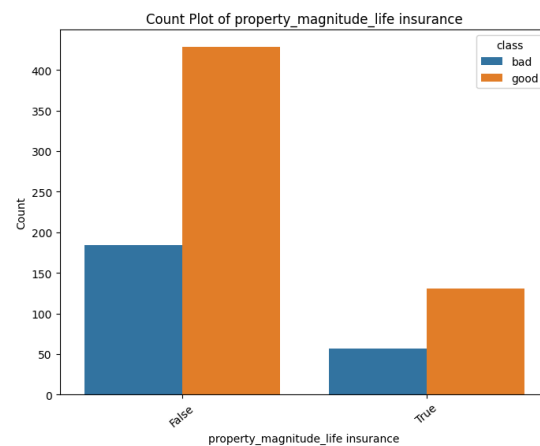
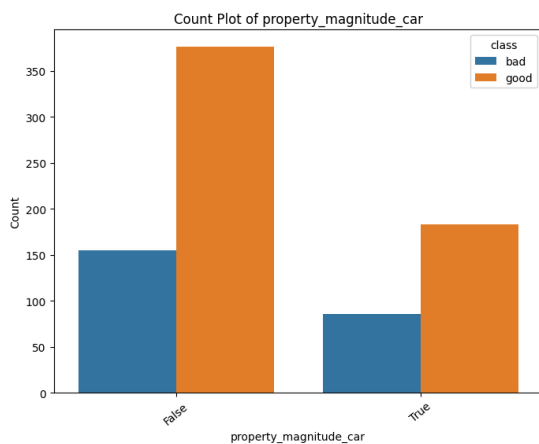
# For numerical features, we are using a box plot to show its relationship
to the target variable
for feature in numerical_features.columns:
    plt.figure(figsize=(8,6))
    sns.boxplot(y=X_train[feature],x=y_train)
    plt.ylabel(feature)
```

```
plt.xlabel("Class")
plt.title(f"Box Plot of {feature}")
plt.show()
```

```
categorical_features = X_train.select_dtypes(exclude=['number'])

for feature in categorical_features.columns:
    plt.figure(figsize=(8,6))
    sns.countplot(x=X_train[feature],hue=y_train)
    plt.xlabel(feature)
    plt.xticks(rotation=40)
    plt.ylabel("Count")
    plt.title(f"Count Plot of {feature}")
    plt.show()
```

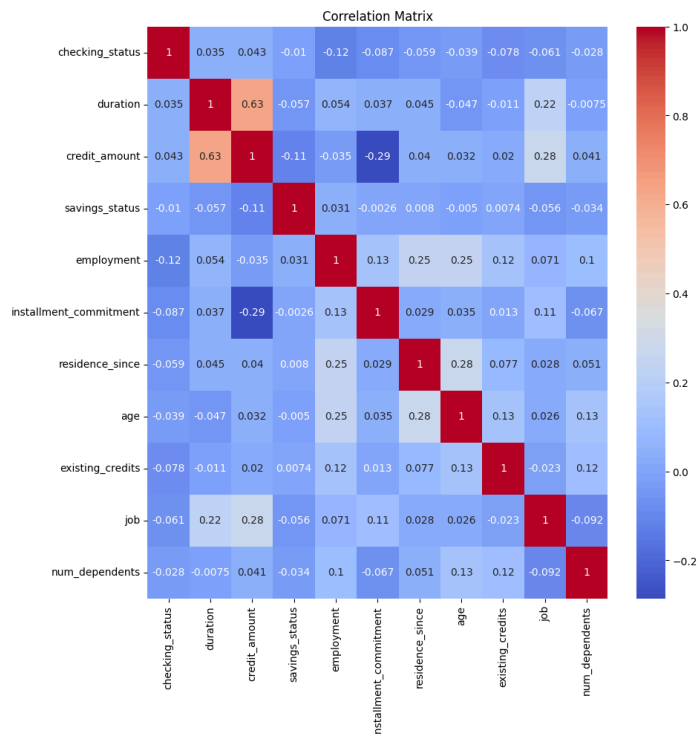
These were the distributions that were essential to our findings:



Key Findings:

- Majority of the people who have life insurance have a class rating of “good”. As a result, owning life insurance is a strong indicator of having minimal/good credit risk.
- Owning a car suggests a higher likelihood of sustained financial stability, as the majority of the people who own a car have a class rating of “good”.
- Classes 0 (bad) and 1(good) for the credit amount box plot have similar medians but varying ranges. Class 1’s range is much tighter than Class 0, indicating more consistency with credit amounts for individuals with good credit risk.

Lastly, we generated a correlation matrix:



From the matrix, credit amount and duration are strongly correlated with a value of 0.63, indicating that the higher the credit amount, the longer it takes for the loan to be repaid.

Additionally, installment commitment and credit amount have a slight negative correlation, meaning that a higher credit amount generally leads to a lower installment commitment.

Supervised Analysis

Overview: Pre-Processing

Three models are chosen for supervised analysis: logistic regression, support vector machine and neural network. Pre-processing procedures are shared between all three models as explained here.

First, the dataset is loaded and converted into a Pandas dataframe. We convert all byte-string entries into strings so they are interpretable. Column names are stripped and cleaned:

```
with open('dataset_31_credit-g.arff', 'r') as file:
    data, metadata = arff.loadarff(file)

# Convert to pandas DataFrame
df = pd.DataFrame(data)

# Convert byte strings to strings in data entries
df = df.map(lambda x: x.decode('utf-8') if isinstance(x, bytes) else x)

# Strip and lowercase all column names for consistency
df.columns = [col.strip().lower() for col in df.columns]
```

Next, we determine which features are numerical and categorical. Among categorical features, we further divide into ordinal (those with natural ordering between possible values) and nominal (without ordering) categories.

Here are all 20 raw input features, categorised:

Table: Types of Raw Input Features

Numerical	Categorical: Ordinal	Categorical: Nominal
<ol style="list-style-type: none">1. duration2. credit_amount3. installment_commitment4. residence_since5. age6. existing_credits7. num_dependents	<ol style="list-style-type: none">1. checking_status2. savings_status3. employment4. job	<ol style="list-style-type: none">1. credit_history2. purpose3. personal_status4. other_parties5. property_magnitude6. other_payment_plans7. housing8. own_telephone9. foreign_worker

One-hot-encoding is applied to nominal categories, where each column gets split into separate columns, each representing a candidate value for this category. True (1) is assigned to a data in the appropriate column, and False (0) for others.

For example, *credit_history* is split into five columns:

1. 'no credits/all paid'
2. 'all paid'
3. 'existing paid'
4. 'delayed previously'
5. 'critical/other existing credit'

A data point with *credit_history* equal to “all paid” would have 1 in that column, and 0 in the others. The major downside to one-hot-encoding is that it significantly increases the number of features. That is why we try to characterize a categorical feature as ordinal unless it can't be.

For ordinal categories, we apply custom mappings for each value to an integer that preserves the natural order. As an example, here is the mapping for the feature *savings_status*:

```
# savings_status order
savings_status_mapping = {
    'no known savings': 0,
    '<100': 1,
    '100<=X<500': 2,
    '500<=X<1000': 3,
    '>=1000': 4
}
df['savings_status'] = df['savings_status'].replace(savings_status_mapping)
```

Third, we apply standard scaling to all features, excluding one-hot-encoded labels. Standardizing data is important so that the values across all features are on a similar scale. This is to prevent those features with large ranges from dominating the model. One-hot encoded labels are excluded as it doesn't have a meaningful scale to speak of; they are just binary labels. Standardization is achieved with sklearn's built-in scaler:

```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(X_to_scale)

# X_to_scale contains only features we choose to standardize
```

The very last step in pre-processing is splitting the dataset. We choose the traditional method of splitting the dataset into three disjoint sets: training, validation and test set, taking up 80%, 10% and 10% respectively. This is purely by a conventional rule-of-thumb. The training set is used to train the model, and the validation set is used for tuning hyperparameters. When the model is finalized, we will test it on the test set to estimate how well our model performs on unseen data.

```
X_train, X_temp, y_train, y_temp = train_test_split(scaled_X, y, test_size=0.2,
random_state=42)
# random_state is to ensure splits are reproducible every time
# its actual value does not matter, only that it's consistent

X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42)
```

Pre-processing is now complete! Next, we will discuss training three different models with this dataset.

Logistic Regression

In logistic regression, a hyperplane is constructed to separate the data into two classes, similar to linear regression. What's different is that the value from the linear combinations of inputs is passed into the sigmoid function, giving us a final output we interpret as the probability of a data point belonging to class 1 (vs class 0). Here is the sigmoid function:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Step 1: Initial Training

We use scikit-learn's model to do our logistic regression. A solver is a specific optimization algorithm that is used under the hood to optimise parameters during model training. The *lbfgs* solver is among the more versatile solvers that scikit offers, as it can support different types of regularization. Here's the code snippet:

```
# use "lbfgs" solver (optimization algo) bc it can support no regularization
unlike other solvers
logreg = LogisticRegression(penalty=None, solver='lbfgs', max_iter=1000)

# train the model on our training set; computes best weights to minimize -ve
log-likelihood function
logreg.fit(X_train, y_train)

# make predictions on test set (to calculate test accuracy)
y_predict = logreg.predict(X_test)

# make predictions on training set (to calculate training accuracy)
y_predict_train = logreg.predict(X_train)
```

Step 2: Initial Evaluation

To help gauge the performance of a model, confusion matrices and receiver operating characteristic (ROC) curves are produced. The structure of a confusion matrix is as follows:

```
# Structure of Confusion Matrix:
#           Predicted
#           0      1
# True  0   [ TN   FP ]
#       1   [ FN   TP ]
```

True and false negatives/positives are shown in the matrix, from which we can extract metrics like precision and recall. Precision is the proportion of true positives among all data that was predicted positive ("good" credit risk). A high precision means that a model is more careful in classifying something as "good". Recall is the proportion of true positives among all data that were actually positive, with a high recall corresponding to a model that is very good at catching "good" cases. Together, they can inform us on how easily a model predicts data as "good" or "bad".

An ROC curve graphs the true-positive and false-positive rates for different thresholds. The threshold is the probability past which the sigmoid output needs to be for a data to be classified as class 1. On the left of the graph, we have lower false positives: being very careful in calling

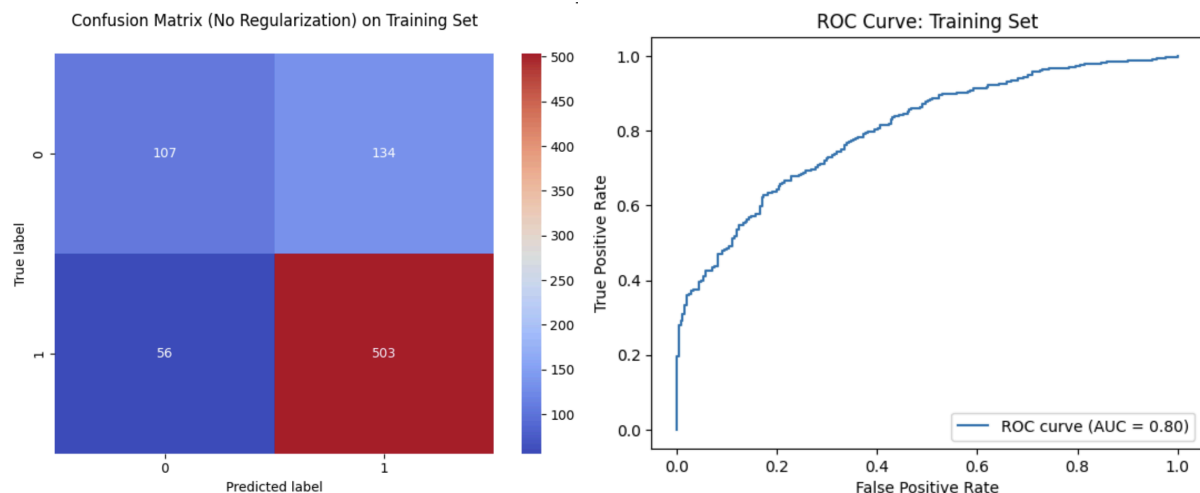
something “good”. This corresponds to having a higher threshold. Because of the higher threshold, we lose some positives and incorrectly classify them as “bad”, explaining the lower true positive rates. As we move to the right of the graph, the threshold lowers, we catch more true positives. But the tradeoff is inevitably classifying more as positives when they aren’t. A perfect classifier would have an ROC that goes from the origin straight to the top-left corner, indicating that the true-positive rate is 100% while having zero false-positives. This in turn corresponds to an area-under-curve (AUC) of 1. The better a classifier, the higher its AUC.

Our initial model is used to predict values for both the training and test test to get an idea of the difference between in-sample and out-of-sample error. Here is a compilation of initial metrics:

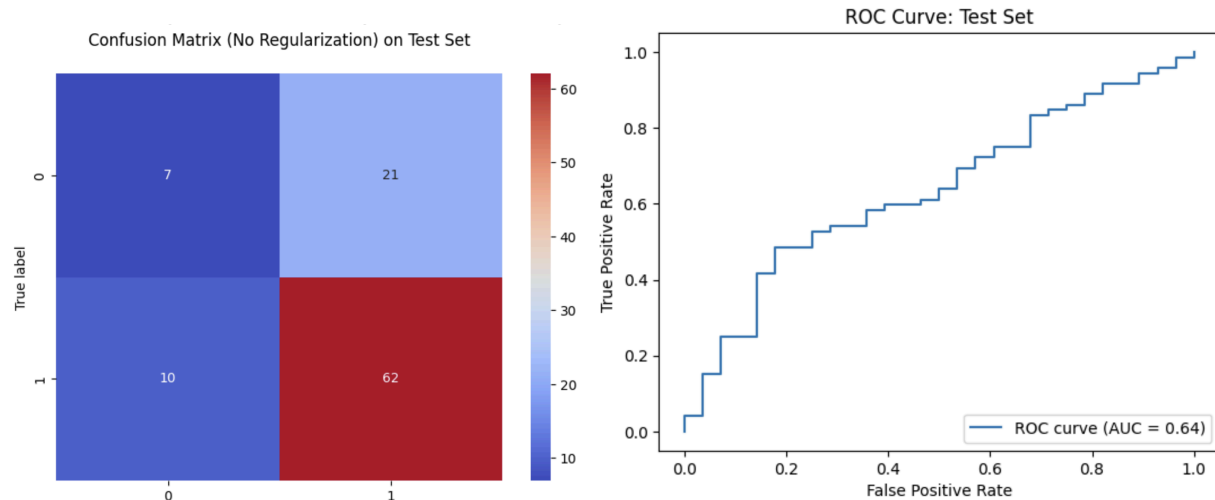
Table: Initial Logistic Model Performance (Training and Test Sets)

Training Set				Test Set			
Precision	Recall	Accuracy	AUC (ROC Curve)	Precision	Recall	Accuracy	AUC (ROC Curve)
0.79	0.9	0.76	0.8	0.75	0.86	0.69	0.64

Graph: Initial Confusion Matrix and ROC Curve on Training Set



Graph: Initial Confusion Matrix and ROC Curve on Test Set



Step 3: Feature Transformations

Next, we seek to improve the initial model through feature transformation. Feature transformations can be used to increase the dimensionality of input features, capturing more complex relationships, though with the possible tradeoff of overfitting the model to training data.

Here, we experiment with three types of feature transformations: polynomial transformation (degree 3), principal component analysis (PCA) and RBF-kernel. For polynomial transformation, degree 3 is chosen because trials with degree 2 produce significantly lower accuracies, while degree 4 produces so many features that it takes very long to compute.

Each feature transformation follows the same procedure: the input data gets transformed using scikit-learn's tools, and then is fed to our logistic model previously built. For example, this is the code for polynomial transformation:

```
# Polynomial feature transformation
poly = PolynomialFeatures(degree=3)
X_train_polynomial = poly.fit_transform(X_train)
X_test_polynomial = poly.transform(X_test)

logreg_polynomial = LogisticRegression(penalty=None, solver='lbfgs',
max_iter=1000)
logreg_polynomial.fit(X_train_polynomial, y_train)
```

Once again, we obtain metrics, confusion matrices and ROC curves for each feature transformation, on both training and test sets:

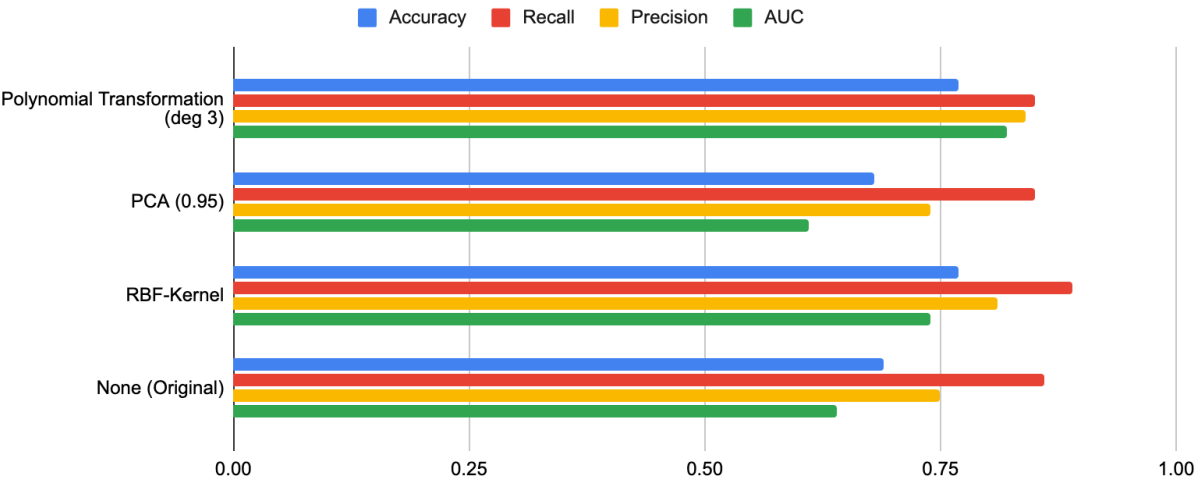
Table: Logistic Model Performance (Training and Validation Sets)
for Different Feature Transformations

	Training Set				Validation Set			
	Precision	Recall	Accuracy	AUC (ROC Curve)	Precision	Recall	Accuracy	AUC (ROC Curve)
Polynomial Transformation (deg 3)	1	1	1	1	0.84	0.85	0.77	0.82
PCA (0.95)	0.76	0.91	0.74	0.77	0.74	0.85	0.68	0.61
RBF-Kernel	1	1	1	1	0.81	0.89	0.77	0.74
None (Original)	0.79	0.9	0.76	0.8	0.75	0.86	0.69	0.64

No regularization applied.

Graph: Logistic Model Performance (Validation Set)
for Different Feature Transformations

Performance on Validation Set: Feature Transformations



Highlighted green shows the best feature transformation. Polynomial transformation is the best among all three because it produces the highest accuracies and AUCs when tested on the

validation set. Although RBF-Kernel results in a perfect in-training accuracy, it does worse when tested on unseen data (validation set), indicating it caused more overfitting to training data than polynomial transformation did.

Step 4: Regularization

We proceed to experiment with different amounts of regularization. Because the data contains so many one-hot-encoded columns, we may face the problem of collinearity. L2 is known to be better than L1 in dealing with collinearity, which is why it's chosen for experimentation here.

We iterate through every regularization strength and get their performance metrics.

```
# Try different values for regularization strength and see how it performs
using * VALIDATION * set
# At the end, choose the best and evaluate model performance on test set
for reg_C in reg_C_vals:
    print("-----")
    print(f"Trying Regularisation Inverse Strength C = {reg_C}.....")
    logreg_regularized = LogisticRegression(penalty='l2', C=reg_C, solver='lbfgs',
max_iter=1000)
    logreg_regularized.fit(X_train, y_train)
    y_predict_regularized = logreg_regularized.predict(X_val)
    y_predict_regularized_train = logreg_regularized.predict(X_train)
    y_predict_prob_reg = logreg_regularized.predict_proba(X_val)[::, 1]
    y_predict_prob_reg_train = logreg_regularized.predict_proba(X_train)[::, 1]

    # Display metrics
    cnf_matrix_regularized = get_confusion_matrix(y_val, y_predict_regularized)
    cnf_matrix_regularized_train = get_confusion_matrix(y_train,
y_predict_regularized_train)
    visualize_confusion_matrix(cnf_matrix_regularized, class_names=[0, 1],
title=f'Confusion Matrix, Reg Strength C={reg_C}:\n Validation Set')
    visualize_confusion_matrix(cnf_matrix_regularized_train, class_names=[0, 1],
title=f'Confusion Matrix, Reg Strength C={reg_C}:\n Training Set')

    print("Metrics on validation set:")
    display_metrics(y_val, y_predict_regularized, cnf_matrix_regularized,
target_names=target_names)
    print("Metrics on training set:")
    display_metrics(y_train, y_predict_regularized_train,
cnf_matrix_regularized_train, target_names=target_names)
    auc = visualize_ROC_plot(y_val, y_predict_prob_reg,
class_mapping=class_mapping, title=f"ROC Curve, Reg Strength C={reg_C}:\n
```

```

Validation Set")
visualize_ROC_plot(y_train, y_predict_prob_reg_train,
class_mapping=class_mapping, title=f"ROC Curve, Reg Strength C={reg_C}:\n
Training Set")
print("-----")

```

Here are the results from testing different L2 regularization strengths:

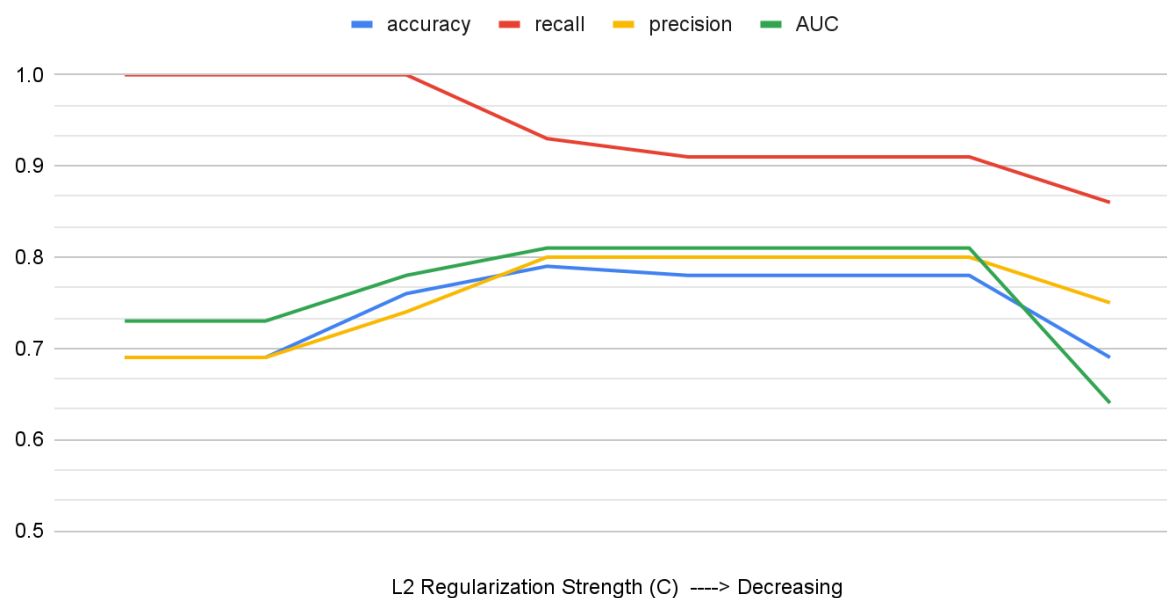
**Table: Logistic Model Performance (Training and Validation Sets)
for Different L2 Regularization Strengths**

C	Training Set				Validation Set			
	Precision	Recall	Accuracy	AUC (ROC Curve)	Precision	Recall	Accuracy	AUC (ROC Curve)
10⁻⁶	0.7	1	0.7	0.73	0.69	1	0.69	0.73
10⁻⁴	0.7	1	0.7	0.74	0.69	1	0.69	0.73
10⁻²	0.72	0.97	0.72	0.77	0.74	1	0.76	0.78
1	0.78	0.91	0.76	0.8	0.8	0.93	0.79	0.81
10²	0.79	0.9	0.76	0.8	0.8	0.91	0.78	0.81
10⁴	0.79	0.9	0.76	0.8	0.8	0.91	0.78	0.81
10⁶	0.79	0.9	0.76	0.8	0.8	0.91	0.78	0.81
None (Original)	0.79	0.9	0.76	0.8	0.75	0.86	0.69	0.64

**Graph: Logistic Model Performance (Training and Validation Sets)
for Different L2 Regularization Strengths**

Performance on Validation Set: Different L2 Regularization Strengths

No feature transformation



The identical metrics for all regularization strengths beyond $C=1$ indicates that adding further regularization is not useful; the behaviour has reached an asymptote. This is why $C=1$ (highlighted green) is taken to be the best.

Step 5: Evaluation on Best Model

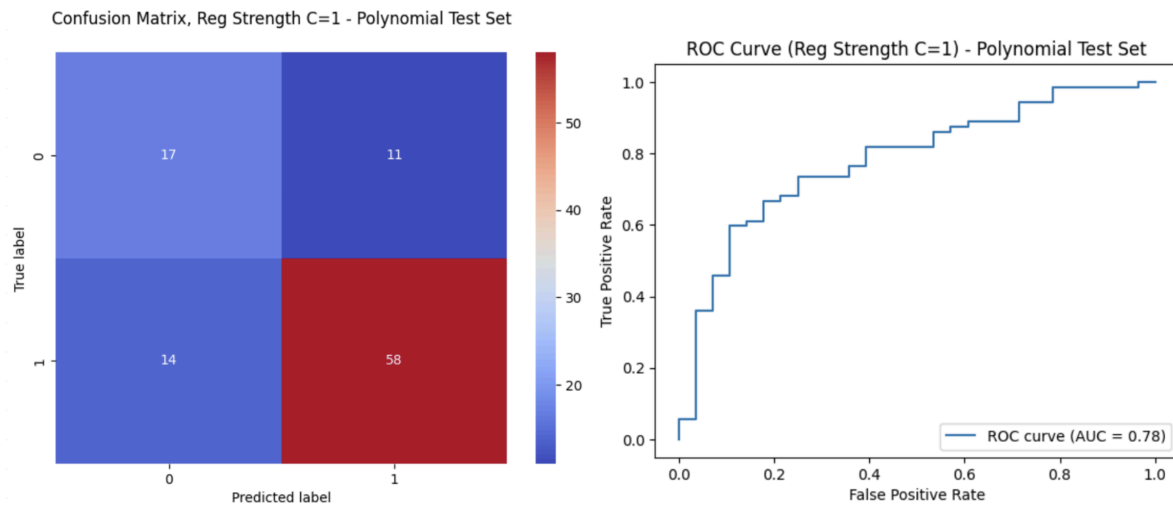
Combining results from feature transformation and regularization strength, we determine that the best model in our experimentation uses polynomial transformation (degree 3) and an L2 regularization strength of 1.

We train the logistic model one last time using these settings, and we produce the following results with the test set:

Table: Final Performance of Best Logistic Model

Test Set			
Precision	Recall	Accuracy	AUC (ROC Curve)
0.84	0.81	0.75	0.78

Graph: Final Confusion Matrix and ROC Curve



The final test accuracy is 75%, which is a reasonably good model.

Support Vector Machine (SVM)

When training the SVM, there were specific parameters that were taken into consideration - accuracy, precision, recall, and hinge loss. This way, we could ensure that we can optimize the model's performance on the training set by maximizing the margin, while also making the model effective in generalizing to new, unseen data.

We did this in several ways. Firstly, we trained the regular SVM using a linear kernel function, after which we applied a polynomial kernel function of degree 3, a radial-basis kernel function, and a PCA transformation. For each feature transformation, it was fitted on the training data, and performed on the training, validation and test data to obtain the aforementioned parameters:

First Row: Linear Kernel (Original)

Second Row: Polynomial Kernel (Degree 3)

Third Row: PCA Transformation

Fourth Row: RBF Kernel

Training Set					Validation Set					Test Set				
Precision	Recall	Accuracy	Error (%)	Hinge Loss	Precision	Recall	Accuracy	Error (%)	Hinge Loss	Precision	Recall	Accuracy	Error (%)	Hinge Loss
0.78	0.9	0.76	24	0.55	0.79	0.96	0.8	20	0.51	0.74	0.89	0.7	30	0.58
0.86	0.98	0.88	12	0.42	0.79	0.96	0.8	20	0.51	0.78	0.94	0.77	23	0.51

0.83	0.99	0.85	15	0.45	0.79	0.97	0.8	20	0.51	0.73	0.92	0.7	30	0.58
0.83	0.99	0.85	15	0.44	0.78	0.97	0.79	21	0.52	0.74	0.9	0.7	30	0.58

The results of the original SVM model showed a relatively low training (24%) and validation error (20%), but a higher error on the test set (30%). This means that the model had a 70% accuracy on the test set, which is generally high; however, in comparison to the training set, there was a small gap. This indicates that our model was slightly overfitting to the training set, making it more ineffective in generalizing to new, unseen data. Furthermore, the hinge loss was above 0.5, indicating that there are some data points that are incorrectly classified/within the margin. As a result, we analyzed the effects of the feature transformations to see if the test error could be improved, while also expanding the boundary.

When analyzing the test set data, the error/accuracy was the same for the PCA and RBF Kernel Transformations. This was due to the fact our original SVM model was also producing a reasonably high accuracy score on the test set (70%); as a result, the model was already generalizing relatively well to new data. Consequently, any transformations would not have been effective.

Additionally, as per the PCA analysis done before, we needed a minimum of 10 features in order to capture nearly all the variance in the dataset. With a high-dimensional dataset consisting of 21 features, applying PCA can cause a loss of important features/data, despite being effective in reducing noise. This can cause the SVM Model to lose valuable information, which will not lead to improvement in model performance.

After analyzing the results, we found that the best transformation was the polynomial kernel of degree 3, as it had the lowest error on the test set and validation set. It was able to decrease the test set error from 30% to 23%, maintain the accuracy on the validation set, and increase the training accuracy from 76% to 88%. Additionally, the hinge loss decreased from 0.58 to 0.51 on the test set, and from 0.55 to 0.42, furthering the model's capability to maximize the margin and perform better on newer data.

Here is the table with the respective data:

Best Transformation:	Polynomial Kernel of Degree 3
Accuracy:	77%
Percentage Error:	23%
Precision:	0.78
Recall:	0.94
Lowest Hinge Loss:	0.51

After that, hyperparameter tuning was performed, where both the regularization strength and the learning rate was manipulated. Below shows the range of values used:

```
learning_rates = [0.001, 0.005, 0.01, 0.05, 0.1, 1, 5]
C_values = [0.01, 0.1, 0.5, 1, 10, 100, 500]
```

For each hyperparameter combination, L1 and L2 regularization were used. We decided to use both L1 and L2 regularization as it would allow for the model to focus on the features that were most important, especially since we had a high-dimensional dataset, and prevent overfitting by penalizing large weights.

At the end, the best parameters were found: the values that minimize hinge loss and percentage error on the validation set, as a way of evaluating the model's performance. The confusion matrix for each case was also generated:

```
# Hyperparameter Tuning with L1 Regularization
best_params = {}
lowest_percentage_error = 100
lowest_hinge_loss = 1
best_pred_hinge = None
best_pred_error = None
for lr in learning_rates:
    for c in C_values:
        sgd_clf = SGDClassifier(loss="hinge", penalty="l1", alpha =
1/(c*len(X_train)), learning_rate="constant", eta0=lr, random_state=42)
        sgd_clf.fit(X_train, y_train)
        y_pred = sgd_clf.predict(X_val)
        curr_hinge_loss = hinge_loss(y_val, y_pred)
        if curr_hinge_loss < lowest_hinge_loss:
            lowest_hinge_loss = curr_hinge_loss
            best_params['hinge_loss'] = (lr, c)
            best_pred_hinge = y_pred
        percentage_error = 100 - accuracy_score(y_pred, y_test) * 100
        if percentage_error < lowest_percentage_error:
            lowest_percentage_error = percentage_error
            best_params['percentage_error'] = (lr, c)
            best_pred_error = y_pred
```

```
# Hyperparameter Tuning with L2 Regularization
lowest_percentage_error = 100
lowest_hinge_loss = 1
```

```

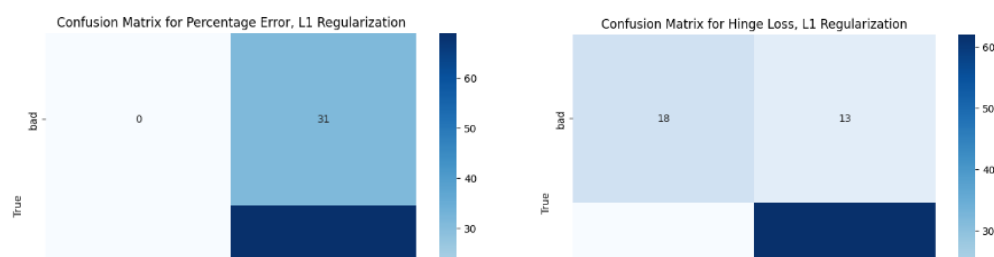
best_params = {}
best_pred_hinge = None
best_pred_error = None
for lr in learning_rates:
    for c in C_values:
        sgd_clf = SGDClassifier(loss="hinge", penalty="l2", alpha =
1/(c*len(X_train)), learning_rate="constant", eta0=lr, random_state=42)
        sgd_clf.fit(X_train, y_train)
        y_pred = sgd_clf.predict(X_val)
        curr_hinge_loss = hinge_loss(y_val, y_pred)
        if curr_hinge_loss < lowest_hinge_loss:
            lowest_hinge_loss = curr_hinge_loss
            best_params['hinge_loss'] = (lr, c)
            best_pred_hinge = y_pred
        percentage_error = 100 - accuracy_score(y_pred, y_test) * 100
        if percentage_error < lowest_percentage_error:
            lowest_percentage_error = percentage_error
            best_params['percentage_error'] = (lr, c)
            best_pred_error = y_pred

```

Below were the results that were obtained:

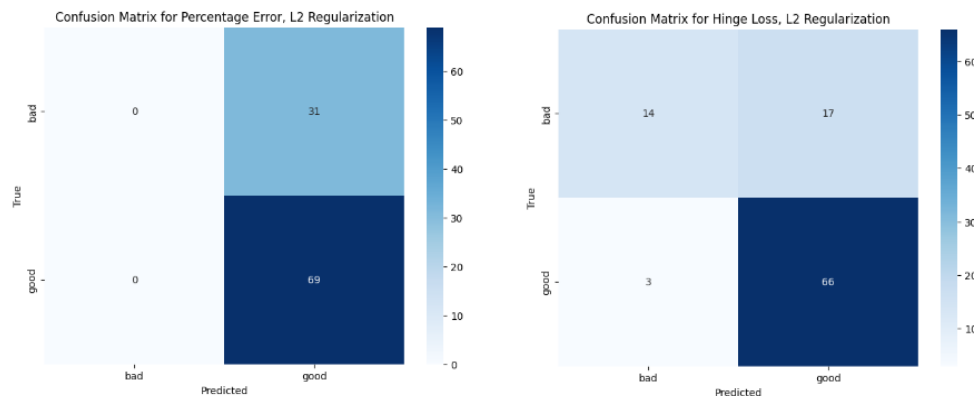
L1 Regularization:

Best Hyperparameters:	Lowest Hinge Loss: 0.51	Percentage Error: 28%
Learning Rate:	0.1	0.001
Regularization Strength (C):	100	0.01
Precision:	0.78	1
Recall:	0.73	0.72



L2 Regularization:

Best Hyperparameters:	Lowest Hinge Loss: 0.51	Percentage Error: 28%
Learning Rate:	0.05	0.001
Regularization Strength (C):	10	0.01
Precision:	0.85	1
Recall:	0.73	0.72

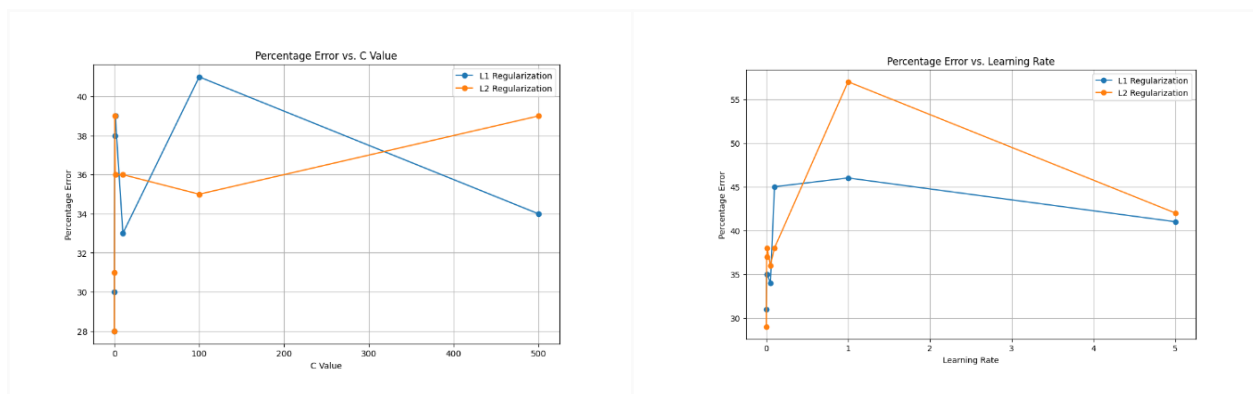


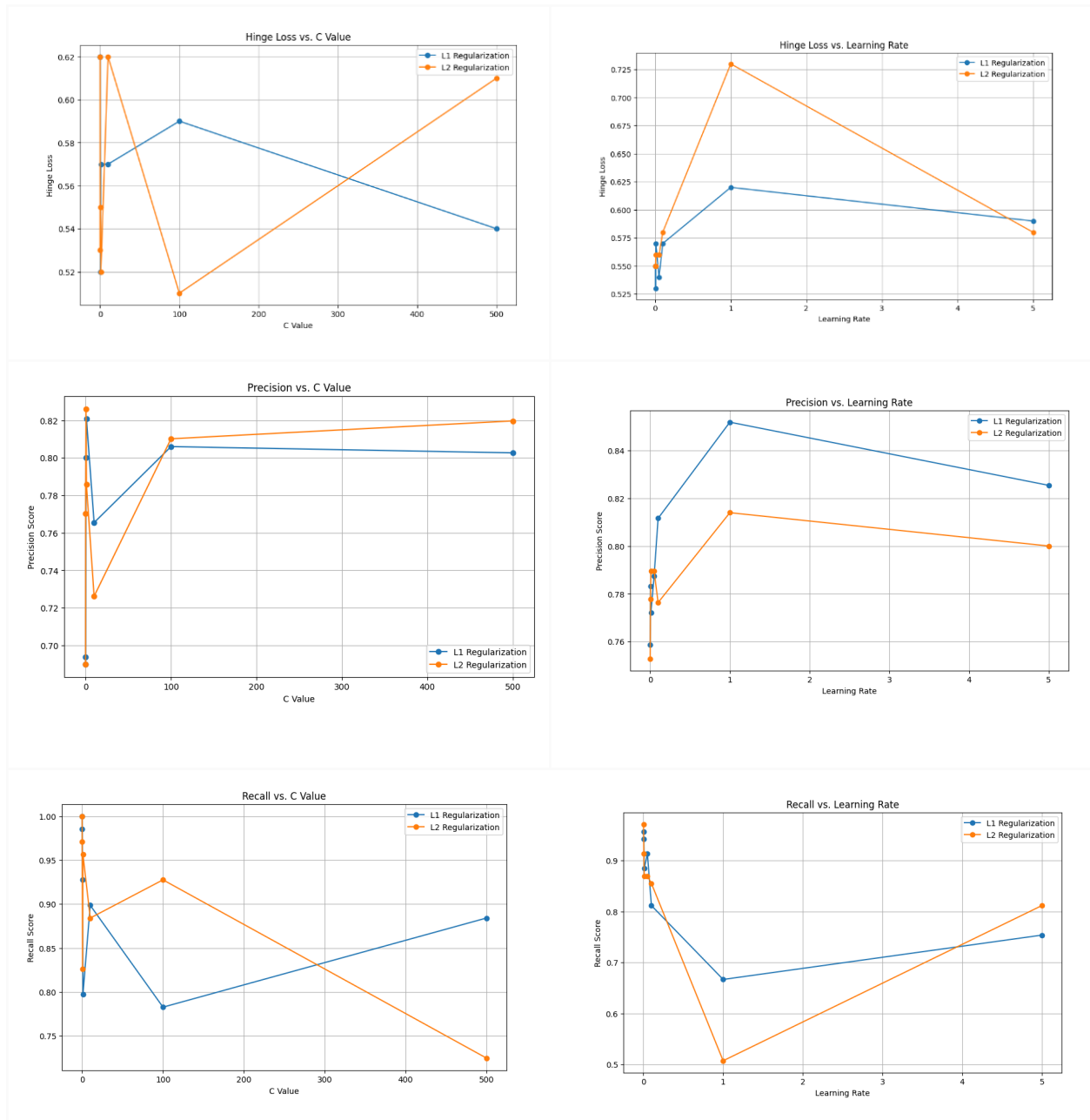
To achieve the lowest hinge loss and maximization of the margin, the learning rate and the regularization strength had to be 0.1 and 100, and 0.05 and 10 for L1 and L2 Regularization, respectively. Compared to the original SVM, it was able to decrease the hinge loss from 0.58 to 0.51. With regularization applied, it prevents overfitting, which penalizes large weights that fit the data too closely. This forces the model to find a more generalized decision boundary, therefore leading to a larger margin. This caused the hinge loss to decrease.

Additionally, to achieve the highest accuracy/lowest percentage error, the learning rate and the regularization strength had to be 0.001 and 0.01 for both L1 and L2 Regularization. Therefore, when it comes to achieving the best model performance, any kind of regularization would yield the same result. Since the original SVM had a reasonably high test accuracy of 70%, it was able

to generalize well to new data; as a result, it did not need a high regularization strength or step size in order to learn the data efficiently. With a small value for the hyperparameters, both regularization techniques were able to increase the model's accuracy from 70% to 72%: a slight improvement in generalization.

After finding the best hyperparameters, a general analysis was conducted to observe the long-term effect of each hyperparameter (learning rate and regularization strength) on validation error and hinge loss for both L1 and L2 Regularization. To analyze the relationship and draw conclusions regarding the model's performance, the graphs below were generated and then analyzed:





Using the graphs from above, we analyzed each performance metric and how it changes with regard to each hyperparameter:

Percentage Error/Accuracy:

- L1 Regularization:**
 - Learning Rate:** Between 0 and 1, the percentage error initially dips and reaches its lowest point, followed by a sharp increase, peaking at approximately 45%. This is attributed to the fact that L1 Regularization is highly sensitive to smaller learning rates early on. Having a learning rate that is too small leads to a small step size, and can therefore take too long to converge. As a result, it can lead to minimal accuracy and high error on the validation set. After reaching its peak at 1, the learning rate continues to increase, and the

percentage error follows a steady decline. This indicates that the model generally adjusts better to new, unseen data with higher learning rates, and can do this efficiently with a larger step size without overshooting. Therefore, the model is able to prevent overfitting most effectively with step sizes that are larger, or step sizes that are small and close to 0. Given the optimal set of hyperparameters (learning rate of 0.001, regularization strength 0.01), it produced a reasonably good model with an accuracy of 72%.

- **C Value:** For smaller values of C, the L1 curve initially experiences a sharp increase, before a sharp dip. The sharp dip indicates that the model is adjusting well to the L1 Regularization, therefore preventing overfitting. However, as C increases, there is a giant spike which leads to a maximum error of above 40%. With such a massive spike, it shows that the model is starting to underfit and is highly sensitive to large C values, therefore causing a massive increase in error. As a result, for smaller values of C, L1 Regularization does an effective job in avoiding the capturing of noise and adjusting well to new patterns in the data.
- **L2 Regularization:**
 - **Learning Rate:** The curve for L2 follows a similar trend to L1 - they both produce the highest accuracy/lowest error quite early on, with a learning rate of 0.001. However, it peaks at a much higher percentage error of above 55% when the learning rate is 1, and follows a steeper decline thereafter. This shows that the L2 Regularization model is more sensitive to the training set than L1. It is more prone to overfit, leading to a higher error on the validation set. However, after a learning rate of 1, the model becomes less sensitive and captures newer, more generalized patterns in the data, therefore explaining the steady decline. Like L1, with the same set of optimal hyperparameters, it produced a model accuracy of 72% as well. As a result, when wanting to improve model performance with regularization, both L1 and L2 regularization would yield the same outcome.
 - **C Value:** The curve initially begins at a low percentage error for very small values of C, therefore exhibiting similar behavior to L1 Regularization. However, it experiences a massive spike in the data, before steadily decreasing as the C Value approaches 100. As the C value increases from 100, the percentage error continues to increase over a long period of time, which suggests that the model could be underfitting. Due to the long-term increase in validation error, the model is not complex enough to capture the patterns in the validation set.

Hinge Loss:

- **L1 Regularization:**
 - **Learning Rate:** Initially, for very small learning rates, L1 Regularization produces a low hinge loss, indicating that the model performs well and is able to generalize well for smaller values of the learning rate. From 0 to 1, it experiences a relatively sharp increase, before slowly declining and becoming flatter for a while. This suggests that as the learning rate increases, the model is able to handle the dataset with less sensitivity, and is more stable with higher learning rates.
 - **C Value:** Initially, the model starts off with a relatively high hinge loss of 0.57, after which it increases to its peak at 0.59, with a high C value of 100. This suggests that the model is highly sensitive to lower values of C, and needs a high regularization strength to perform

well and adjust to new, unseen data. After C increases from 100, the hinge loss slowly decreases, suggesting that the model is effective and can generalize better with a higher rate of regularization.

- **L2 Regularization:**

- **Learning Rate:** The curve follows a similar pattern to L1; however, it increases at a much steeper rate, reaching a significantly higher hinge loss of above 0.725 at a learning rate of 1. Additionally, the decline is much sharper after that, relative to L1. Therefore, while L2 is able to perform well for very small learning rates, and slowly adjust to higher learning rates, it is a model that is more unstable and sensitive to the learning rate.
- **C Value:** Initially, the model starts off with a very high hinge loss of 0.62, and dips sharply, followed by a massive spike to a hinge loss of 0.62 again. This suggests that the model is highly sensitive to very small C values, like L1. However, as C increases to 100, L2 significantly decreases to reach its lowest point, suggesting that a regularization strength of C is optimal in order for the model to generalize well. After that point, the hinge loss consistently increases long-term, as the C value approaches 500. Due to the consistent, and vast increase in hinge-loss, it suggests that too much regularization was performed. As a result, the model became too simple to capture unseen patterns in the data.

Precision:

- **L1 Regularization:**

- **Learning Rate:** Initially, for very small learning rates (0 to 1), there was a massive spike in the precision, suggesting that with a smaller step size, the model is taking a more cautious, slow approach. Because of this, the risk of overshooting is minimized, and it leads to fewer false positive predictions in the dataset. As a result, at a learning rate of 1, it reaches its optimal precision of above 0.84. However, after the learning rate increases from 1 to 5, the precision steadily declines, which can be attributed to too big of a step size: by having such a high learning rate, the model will end up diverging. This causes the model to miss important patterns in the dataset, leading to more points being incorrectly classified. As a result, the number of false positive predictions will increase, and the precision score goes down. This suggests that L1 Regularization will produce optimal precision with a small step size/learning rate.
- **C Value:** The behavior of the L1 curve was slightly similar to L2; however, when undergoing a dip and an increase from C = 0 to C = 100, it was not as sharp as L2. During that period, it generally had a higher precision score; therefore, if a smaller amount of regularization is done, precision and performance is better when undergoing L1 Regularization. However, at C = 100, and when C continues to increase thereafter, the precision score is lower than L2, and it remains that way long-term. In terms of its individual behavior, it slowly decreases, but the change is quite minimal. During this period, the precision score remains above 0.8, which is quite high. As a result, preventing overfitting through regularization doesn't significantly impact the precision, suggesting that the model would have already been performing generally well on new data.

- **L2 Regularization:**

- **Learning Rate:** The curve follows a similar pattern to L1's curve; however, it does not have as big of an increase for smaller learning rates, the maximum precision score is generally not as high, and it does not decline as steadily when the learning rate increases from 1 to 5. As a result, having a small learning rate is also ideal for L2 Regularization; however, the

precision score will not be affected as much and has a lower sensitivity, and it will still produce more false positives than it would if it underwent L1 Regularization. Therefore, in order to maximize precision, it is ideal to use L1 Regularization, and use a small learning rate.

- **C Value:** As C increases from 0 to 100, the curve makes a sharp increase to its maximum precision of above 0.82, followed by a sharp dip, and then another massive increase to a precision of approximately 0.81, at a C value of 100. This suggests that the model is highly sensitive to smaller regularization strengths. However, as the regularization strength increases from C = 100 to C = 500, the precision slowly and steadily increases, though not a significant amount. Therefore, similar to L1, after a certain point, preventing overfitting doesn't significantly impact the precision, meaning that the model could have already been adjusting generally well to complex patterns in the data.

Recall:

- L1 Regularization:

- **Learning Rate:** The recall significantly decreases as the learning rate increases from 0 to 1. This could be due to a very small step size, which makes the process of learning for the model quite slow, and therefore inefficient in progressing towards better performance. At LR = 1, recall reaches its lowest value of approximately 0.65, before slowly increasing as the learning rate increases from 1 to 5. This behavior can be explained by the fact that since there is a bigger step size, there is increased efficiency in learning the model. This leads to more correct classification of points, and fewer false negatives being predicted.
- **C Value:** Between C = 0 and C = 100, the recall significantly decreases, then increases, followed by a sharp decrease to its lowest value at C = 100, with a recall of approximately 0.78. This can be attributed to L1 regularization not being strong enough - it has a higher tendency to overfit, and doesn't generalize well to unseen data. This can lead to more false negative predictions, and overall misclassified points, and a lower recall overall. However, after C = 100, it steadily increases up till C = 500, which shows that with increased regularization strength, the model has become simple enough to generalize well to new data, and have a higher likelihood of making correct predictions, and less false negatives. Therefore, L1 Regularization works effectively with high regularization strength.

- L2 Regularization:

- **Learning Rate:** The behavior of this curve is quite similar to L1; however, it generally had more unstable performance - when the learning rate was increasing from 0 to 1, it had a much bigger dip. However, after the learning rate was increasing from 1 to 5, it increased at a much steadier rate, and produced higher recall values at approximately 3.8. Therefore, using L2 regularization leads to higher sensitivity to the step size. When it comes to a learning rate between 1 and 4, L1 Regularization performs better; however, after a certain point, the model with L2 is able to learn more efficiently, and therefore produce improved classification results.
- **C Value:** The behavior of the L2 curve was quite different from L1; Despite a few sharp dips and increases from C = 0 to C = 100, it generally maintained a high recall score of above 0.90 by the time C increased to 100. This shows that for smaller C values, L2 Regularization performs well, and has a higher likelihood of predicting positive/true

instances. However, after $C = 100$, the recall significantly decreases to its lowest value of below 0.75. This suggests that there was too much regularization - due to a high regularization strength, the model became too simple, causing it to poorly perform on both the validation and training set. As a result, with larger C values, the model starts to underfit. This indicates that the original model, despite slightly overfitting, does not need that much L2 Regularization in order to generalize well to unseen data.

Neural Network

Here, a deep neural network is chosen. Every hidden layer has activation nodes that take the weighted sum of inputs, pass it through an activation function and deliver its output. The *sigmoid* function is used as the activation function, where z represents the weighted sum of inputs:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Convolutional neural networks are not chosen because our dataset does not consist of image or video inputs. Recurrent neural networks are not picked because our data is not made of sequential data like sentences.

Step 1: Training

We begin by creating a new classifier with scikit-learn's *MLPClassifier* (multi-layer), important as we'll be experimenting with different amounts of hidden layers. The initial neural network has 2 hidden layers

```
# Use MLPClassifier with the sigmoid activation function
hidden_layers_amounts = [2, 10, 15, 30]
num_hidden_layers = hidden_layers_amounts[0]

model = MLPClassifier(hidden_layer_sizes=(num_hidden_layers,),
activation='logistic', solver='adam', max_iter=1000)
model.fit(X_train, y_train)

y_predict = model.predict(X_test)
y_predict_train = model.predict(X_train)
```

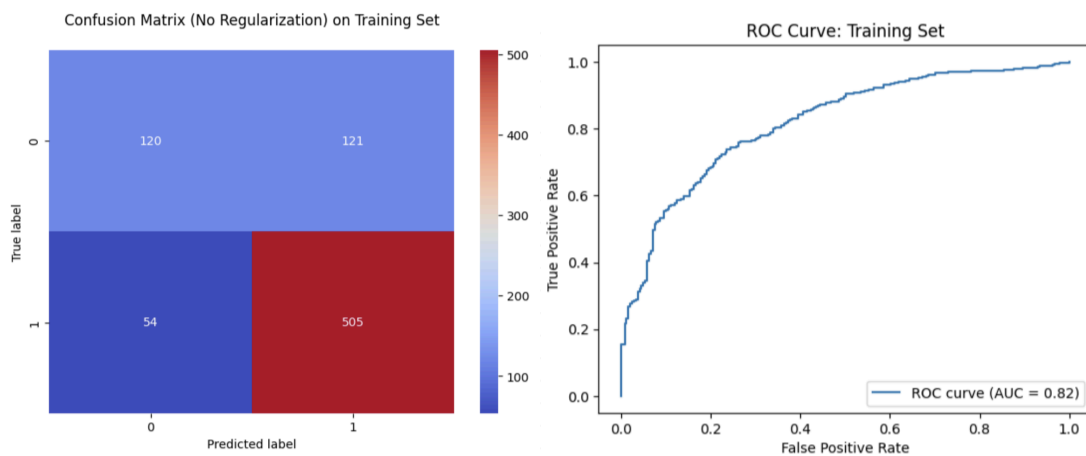
Step 2: Initial Evaluation

After testing our initial neural network on the test set and training set, we obtain these metrics:

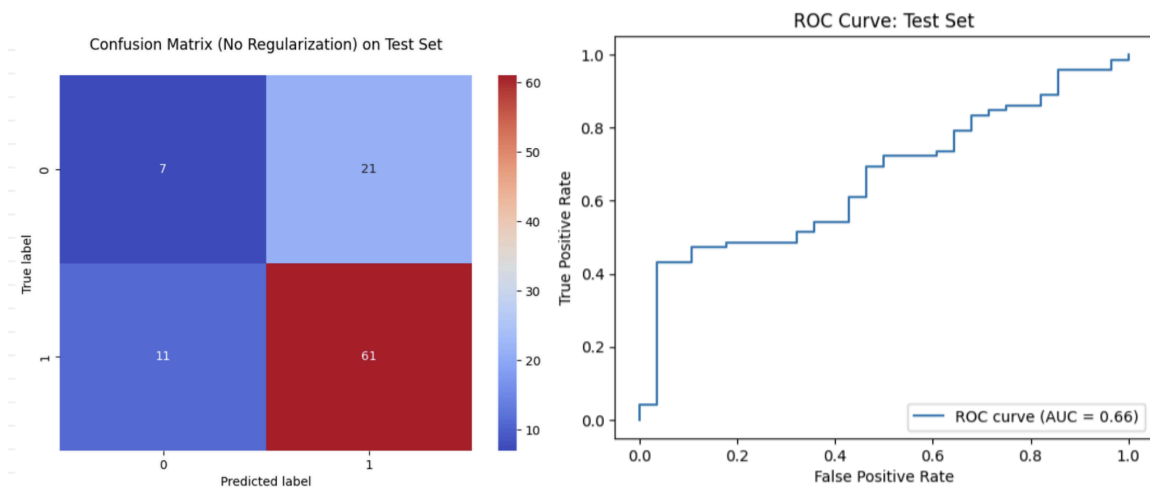
Table: Performance of Initial Neural Network (Training and Test Sets)

Training Set				Test Set			
Precision	Recall	Accuracy	AUC (ROC Curve)	Precision	Recall	Accuracy	AUC (ROC Curve)
0.81	0.9	0.78	0.82	0.74	0.85	0.68	0.66

Graph: Performance of Initial Neural Network (Training Set)



Graph: Performance of Initial Neural Network (Test Set)



Both the accuracy and AUC on the ROC curve are higher on the training set than the test set. This is expected as our model was fitted using the training set. The test set's AUC is at 0.66, which is only marginally better than random guessing (AUC = 0.5). By experimenting with

different amounts of hidden layers next, we hope to find a model that performs better on the test set.

Step 3: Number of Hidden Layers in the Neural Network

We'll be experimenting with the following amounts of hidden layers:

```
hidden_layers_amounts = [2, 10, 15, 30]
```

Adding more layers allows us to capture more complex relationships in the training data. But adding too many layers comes with the potential pitfall of overfitting to the training data, resulting in a poor out-of-sample accuracy. Testing 2, 10, 15 and 30 hidden layers covers a fairly wide range, while remaining reasonably computable on our computers.

After training our neural network using these different amount of layers, we get the following metrics:

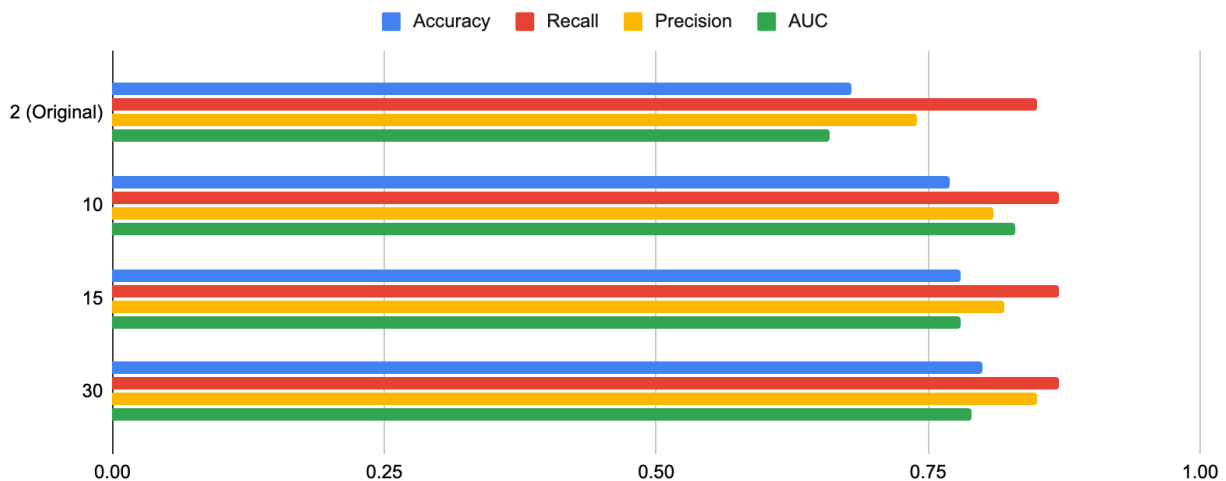
**Table: Performance of Neural Network (Training and Validation Sets)
Using Different Amounts of Hidden Layers**

	Training Set				Validation Set			
	Precision	Recall	Accuracy	AUC (ROC Curve)	Precision	Recall	Accuracy	AUC (ROC Curve)
2 (Original)	0.81	0.9	0.78	0.82	0.74	0.85	0.68	0.66
10	0.83	0.92	0.83	0.88	0.81	0.87	0.77	0.83
15	0.87	0.94	0.86	0.92	0.82	0.87	0.78	0.78
30	0.92	0.96	0.91	0.96	0.85	0.87	0.8	0.79

No regularization applied.

**Graph: Performance of Neural Network (Training and Validation Sets)
Using Different Amounts of Hidden Layers**

Performance on Validation Set: Amount of Hidden Layers



Using 10 hidden layers (highlighted green) provides the best accuracy and AUC both in-sample and out-of-sample. Although using 15 hidden layers results in a better accuracy for validation set by 1%, 10-hidden layers has a greater edge in AUC on validation set, which is why it has been interpreted as the best.

Why does using more than 10 hidden layers result in worse performance? This likely has to do with overfitting training data. With more hidden layers, more complex relationships within training data are captured, much of which may be noise. From the table, using 30 hidden layers results in an extremely high in-sample accuracy, while out-of-sample accuracy continues to drop. This is a clear indication that overfitting occurred.

Step 4: Regularization

Using 10 hidden layers, we continue to test different amounts of regularization. As in logistic regression, we choose to experiment with different L2 strengths:

```
reg_strengths = [ 10**x for x in range(-6,7,2) ]
```

For each `reg_strength`, we train our neural network on them and test the model on both the validation set and training set:

```
model_regularized = MLPClassifier(hidden_layer_sizes=(10,),  
alpha=reg_strength, activation='logistic', solver='adam', max_iter=1000)
```

```

model_regularized.fit(X_train, y_train)
y_predict_regularized = model_regularized.predict(X_val)
y_predict_regularized_train = model_regularized.predict(X_train)

```

Here is the performance for each L2 regularization strength:

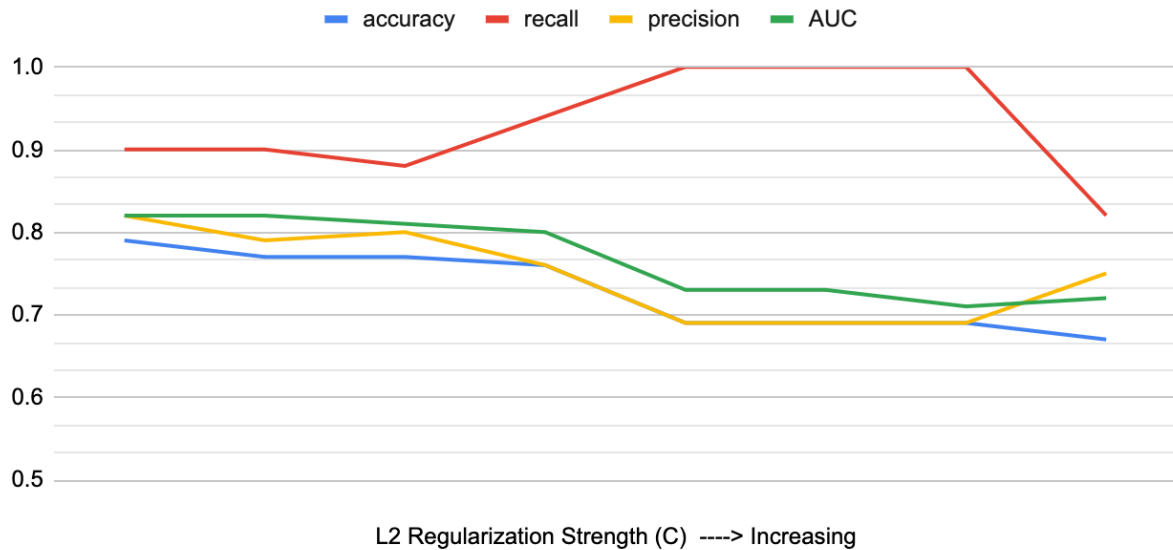
**Table: Performance of Neural Network (Training and Validation Sets)
Using Different L2 Regularization Strengths**

	Training Set				Validation Set			
	Precision	Recall	Accuracy	AUC (ROC Curve)	Precision	Recall	Accuracy	AUC (ROC Curve)
10⁻⁶	0.86	0.92	0.84	0.9	0.82	0.9	0.79	0.82
10⁻⁴	0.85	0.93	0.84	0.89	0.79	0.9	0.77	0.82
10⁻²	0.85	0.94	0.84	0.9	0.8	0.88	0.77	0.81
1	0.76	0.93	0.76	0.79	0.76	0.94	0.76	0.8
10²	0.7	1	0.7	0.74	0.69	1	0.69	0.73
10⁴	0.7	1	0.7	0.73	0.69	1	0.69	0.73
10⁶	0.7	1	0.7	0.72	0.69	1	0.69	0.71
None	0.86	0.92	0.84	0.91	0.75	0.82	0.67	0.72

**Graph: Performance of Neural Network (Training and Validation Sets)
Using Different L2 Regularization Strengths**

Performance on Validation Set: Different L2 Regularization Strengths

No feature transformation



Applying L2 regularization strength of 10^{-6} yields both the best in-sample and out-of-sample accuracies. This represents applying a minimal amount of regularization, informing us that the model benefits very little from regularization. If anything, this goes to support that using 10 hidden layers is a suitable choice for the data and/or that the training data is not too noisy. In other words, the neural network was not overfitted to the training data.

Step 5: Different learning rates

To continue experimenting, we try different learning rates used in the optimizer:

```
learning_rates = [ 10**x for x in range(-5, 8, 2) ]
```

Building on previous findings, we use 10 hidden layers and apply L2 regularization of 10^{-6} .

Here are the results:

**Table: Performance of Neural Network (Training and Validation Sets)
Using Different Learning Rates**

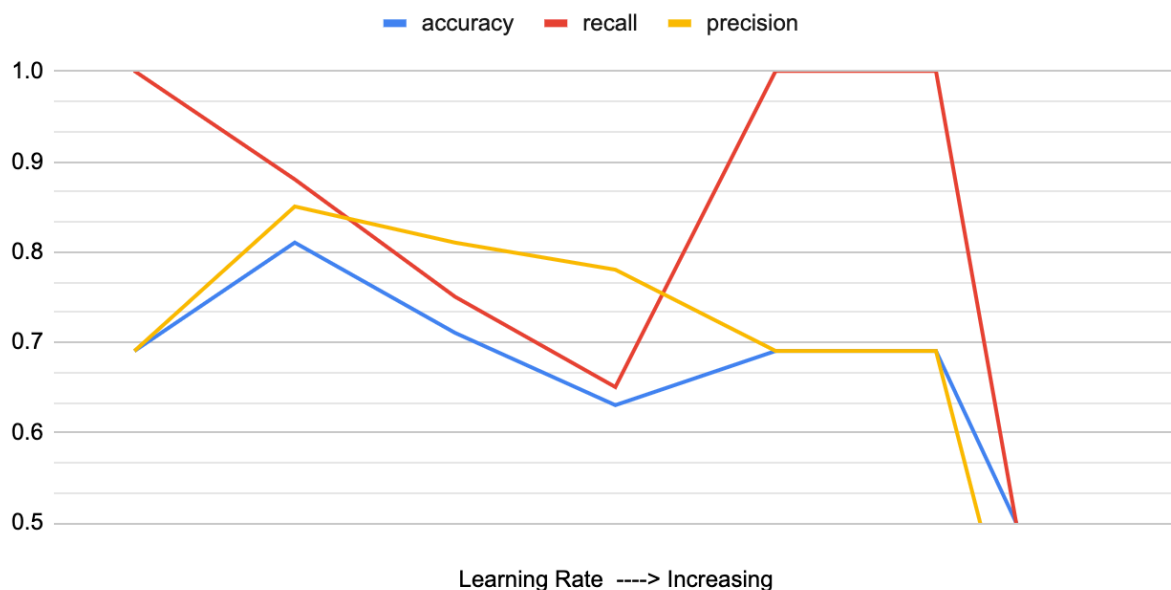
	Training Set	Validation Set
--	--------------	----------------

	Precision	Recall	Accuracy	AUC (ROC Curve)	Precision	Recall	Accuracy	AUC (ROC Curve)
10^{-5}	0.7	1	0.7	0.49	0.69	1	0.69	0.49
10^{-3}	0.86	0.93	0.85	0.9	0.85	0.88	0.81	0.84
10^{-1}	0.99	0.99	0.99	1	0.81	0.75	0.71	0.75
10	0.85	0.67	0.69	0.73	0.78	0.65	0.63	0.67
10^3	0.7	1	0.7	0.5	0.69	1	0.69	0.5
10^5	0.7	1	0.7	0.5	0.69	1	0.69	0.5
10^7	0	0	0.3	0.5	0	0	0.31	0.5

**Graph: Performance of Neural Network (Training and Validation Sets)
Using Different Learning Rates**

Performance on Validation Set: Different Learning Rates

No feature transformation



A learning rate of 10^{-3} is found to be the best learning rate here. From the table, the lower learning rate of 10^{-5} performs significantly worse because the model is updating parameters way too slowly given the same number of iterations, thereby struggling to fit to the training data, as evident from the extremely poor in-sample accuracy. For learning rates higher than 10^{-3} , we see decreasing performance on the validation set. Taking steps too large during gradient

descent may cause erratic behaviour, not being able to settle on a final model that can generalize well to unseen data.

Step 6: Evaluation on Best Model

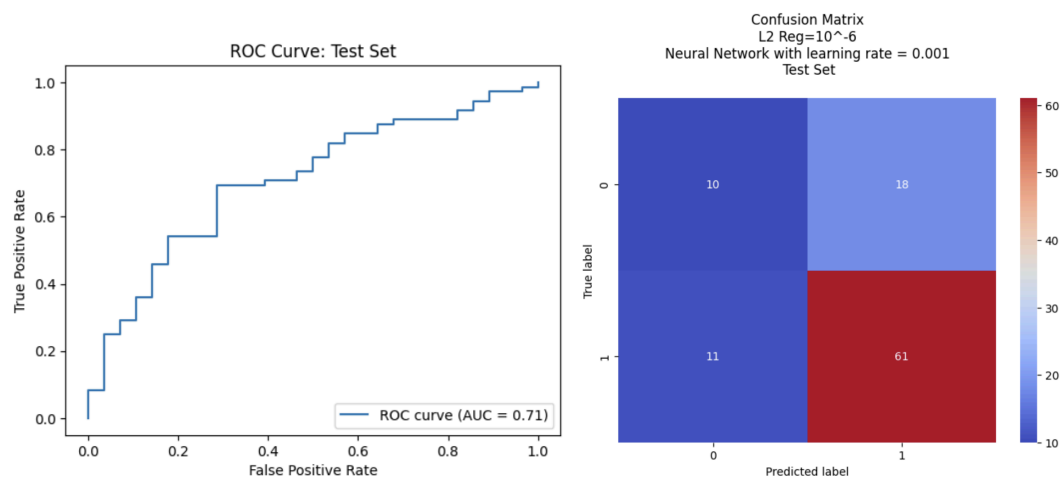
From the previous steps, we find that the optimal neural network - out of the configurations we tested - has 10 hidden layers, L2 regularization strength of 10^{-6} and a learning rate of 0.001. These are found by testing on the validation set. Now, we use these configurations to train our final model and test on the testing set.

Here is the final performance:

Table: Final Performance (Training and Test Sets)

Training Set				Test Set			
Precision	Recall	Accuracy	AUC (ROC Curve)	Precision	Recall	Accuracy	AUC (ROC Curve)
0.86	0.94	0.85	0.92	0.77	0.85	0.71	0.71

Graph: Performance of Final Neural Network (Test Set)



The final test accuracy sits at 71%, which is a reasonably good model.

Table of Results

[Attached here](#) is the link to all of the project data and results that were generated. All of our analysis of the data is present in each specific section, per model.

Analytical Discussion

Key Observations for Each Model

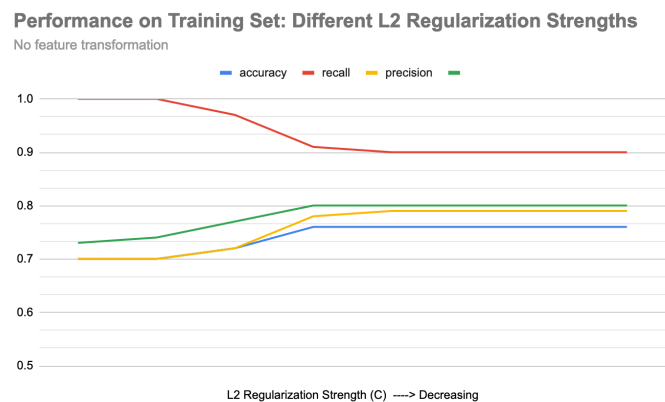
Logistic

- Polynomial transformation of degree 3 results in 12% higher validation accuracy than no transformation
 - Results in 12% higher validation accuracy than applying no transformation
 - **Feature Dimensionality:** This suggests that the dataset likely has non-linear relationships that are better captured when features are transformed to a higher dimension.
 - In comparison with PCA:
 - **Bias-Variance Tradeoff:** PCA causes significantly less overfitting to training data. However, it performs the worst on the validation set. If anything, this suggests that using PCA resulted in an underfitting of data: the model was not able to capture the complex relationships in the training data that it also fails to generalize to unseen data.
 - In comparison with RBF-Kernel:
 - **Bias-Variance Tradeoff:** RBF-Kernel transformation results in overfitting to training data as seen in the table below, where the in-sample accuracy reaches a perfect one.

**Table: Logistic Model Performance (Training and Validation Sets)
for Different Feature Transformations**

	Training Set				Validation Set			
	Precision	Recall	Accuracy	AUC (ROC Curve)	Precision	Recall	Accuracy	AUC (ROC Curve)
Polynomial Transformation (deg 3)	1	1	1	1	0.84	0.85	0.77	0.82
PCA (0.95)	0.76	0.91	0.74	0.77	0.74	0.85	0.68	0.61
RBF-Kernel	1	1	1	1	0.81	0.89	0.77	0.74
None (Original)	0.79	0.9	0.76	0.8	0.75	0.86	0.69	0.64

- Where C is the inverse regularization strength, C=1 is best when tested on the validation set
 - Lower values of C (more regularization) produce worse validation accuracies
 - Higher values of C (less regularization) produces no change in performance to C=1
 - **Bias-Variance Tradeoff:** If applying more regularization leads to worse performance, then it indicates that the model may be underfitted to the training data. This conclusion can be clearly illustrated by analysing the accuracy (or AUC) on the training set. For the training set, its accuracy and AUC increase as we lessen the amount of regularization. This means that too much regularization applied was preventing relationships being captured within the training data. As we decrease regularization to C=1, the training set's accuracy plateaus and sees no further improvement.



SVM

Key Findings:

- Using a polynomial kernel of degree 3 improved the model's generalization ability, while the other transformations did not make a difference.

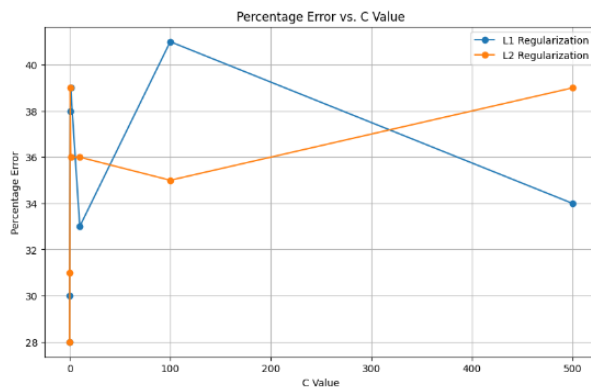
Training Set					Validation Set					Test Set				
Precision	Recall	Accuracy	Error (%)	Hinge Loss	Precision	Recall	Accuracy	Error (%)	Hinge Loss	Precision	Recall	Accuracy	Error (%)	Hinge Loss
0.78	0.9	0.76	24	0.55	0.79	0.96	0.8	20	0.51	0.74	0.89	0.7	30	0.58
0.86	0.98	0.88	12	0.42	0.79	0.96	0.8	20	0.51	0.78	0.94	0.77	23	0.51
0.83	0.99	0.85	15	0.45	0.79	0.97	0.8	20	0.51	0.73	0.92	0.7	30	0.58
0.83	0.99	0.85	15	0.44	0.78	0.97	0.79	21	0.52	0.74	0.9	0.7	30	0.58

The original SVM had a reasonably high performance of 70% on the test set; as a result, it already had the ability to generalize relatively well to new, unseen data. However, due to the

training accuracy being slightly higher, the model did slightly overfit the data. It was still able to capture complex patterns as opposed to noise, which is why when performing feature transformations like PCA or RBF, it did not make a significant difference.

As a result, Polynomial Kernels, of a relatively low degree of 3, provided a good balance between allowing the SVM model to generalize better, without becoming too simple to underfit the training data.

- The model's generalization ability improves with a small amount of regularization. A high regularization strength will cause the model to become too simple and underfit.



For L2 Regularization, increasing C past 100 caused a consistent, long-term increase in percentage error until C = 500. Due to this long-term increase, the model became too simple and failed to capture complex patterns, therefore leading to poor performance on the validation set.

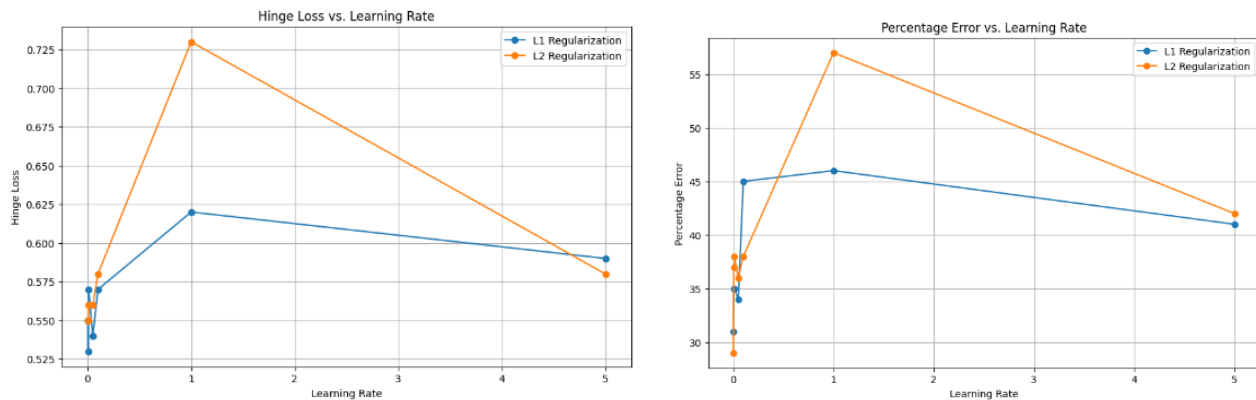
Despite the fact that L1 Regularization increased with large values of C, it generally produced a much larger error than it did for regularization strengths that were close to 0. Therefore, achieving minimal error and highest accuracy calls for small regularization strengths.

Choosing the value of C plays a key role in the **bias-variance** tradeoff. Increasing the value of C leads to further regularization, and a lower variance. However, the model becomes simpler, and leads to more underfitting, which increases the bias. This causes the model to perform poorly on both the training set and validation set. On the other hand, having a smaller value of C creates a more complex model with greater variance and lower bias. This leads to an increased likelihood of overfitting, causing great performance on the training set, but poor performance on the validation and test set.

The original SVM model does overfit, since it performs better on the training set and validation set than it does on the test set. As a result, it has a lower bias and a higher variance. However, given the reasonably high accuracy on the test set (70%), the variance isn't too high, and the value of C for regularization does not need to be too high, but slightly smaller. That way the

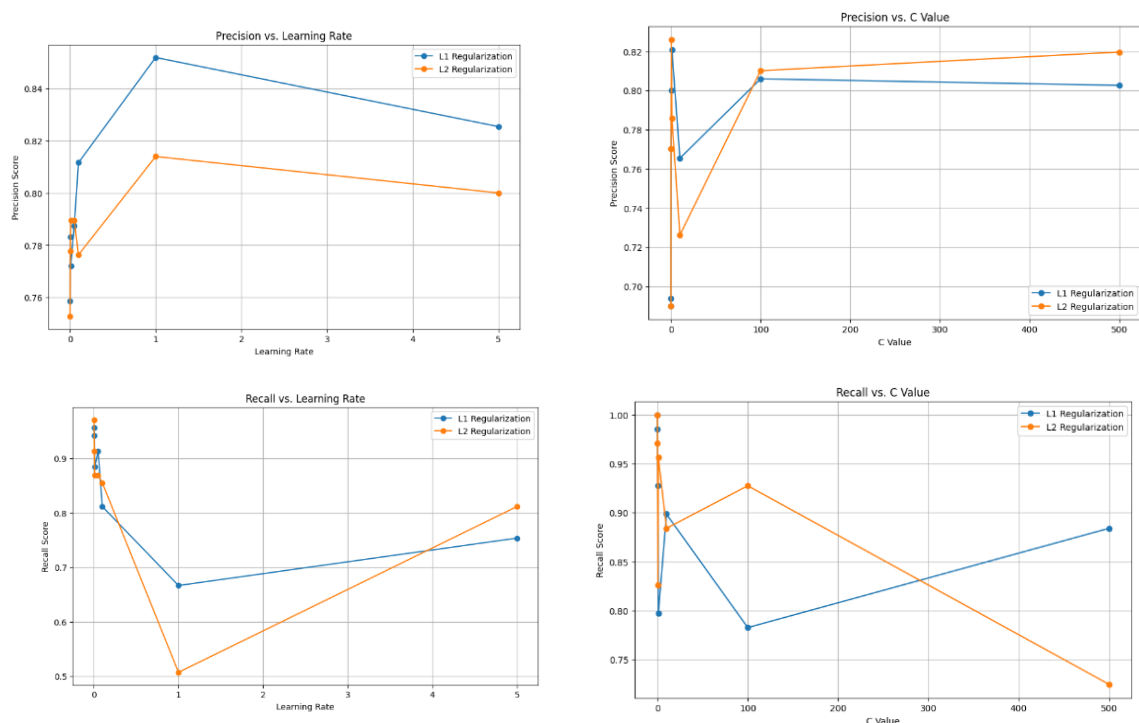
variance can be slightly decreased, and as a result, the bias can be increased, in order to have a better generalization to new data.

- The ideal hyperparameter values, both for L1 and L2 Regularization and to maximize accuracy and hinge loss, is a learning rate of 0.001 and a regularization strength of 0.01.



As explained previously, a small regularization is needed in order to prevent the model from underfitting. As for the learning rate, though a large learning rate causes a decrease in hinge loss and percentage error, they are generally larger than the values produced for much smaller learning rates that were close to 0. Therefore, having a small step size is ideal for minimizing hinge loss and error. That way, it prevents the risk of overshooting, and allows the model to efficiently learn and capture complex patterns in unseen data.

- In order to maximize precision, a small learning rate and a large regularization strength is ideal. However, in order to maximize recall, a large learning rate and smaller regularization strength is needed.



To maximize precision, a smaller learning rate is preferred as it lowers the risk of overshooting, and the model can progress cautiously to learn the data effectively. That way, it can lead to more precise decision boundaries, and a smaller likelihood of predicting false positives. Additionally, a greater C value is preferred as it leads to more regularization, less overfitting, and a better generalization ability. That way, the model can better capture complex patterns and features in new, unseen data, leading to improved classification overall. As a result, it will have less tendency to predict false positives.

To maximize recall, a large learning rate is preferred as the model converges more quickly, allowing it to capture more positives. However, that does also include false positives, thus leading to a lower precision. Additionally, a lower value of C is preferred as with less regularization, the model is able to work with more complex decision boundaries, which, by the same token, also can include false positives. Therefore, a greater recall can lead to lower precision.

Neural Network

- 10 hidden layers
 - Performed the best out of 2 vs 10 vs 15 vs 30 hidden layers
 - **Bias-Variance Tradeoff:** From the results, 10 hidden layers strikes the best balance between capturing complexity in the data while not overfitting the training data. With fewer hidden layers (2), the in-training accuracy is lower, which suggests the inability to capture data relationships. This inability translates to a poor generalization on unseen data, evident from the low validation accuracy. With more hidden layers like 15 and even 30, in-training accuracy becomes close to perfect. However, too many layers caused overfitting, which is why when tested on the validation set, the accuracy becomes lower, indicating poor generalization.

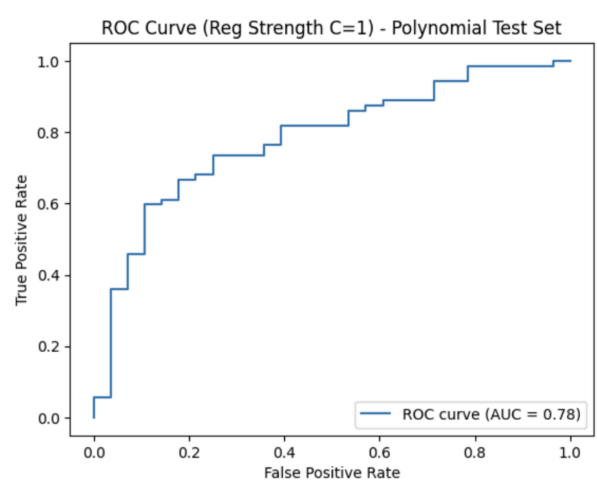
Hidden Layers	Training Set				Validation Set			
	Precision	Recall	Accuracy	AUC (ROC Curve)	Precision	Recall	Accuracy	AUC (ROC Curve)
2 (Original)	0.81	0.9	0.78	0.82	0.74	0.85	0.68	0.66
10	0.83	0.92	0.83	0.88	0.81	0.87	0.77	0.83
15	0.87	0.94	0.86	0.92	0.82	0.87	0.78	0.78
30	0.92	0.96	0.91	0.96	0.85	0.87	0.8	0.79

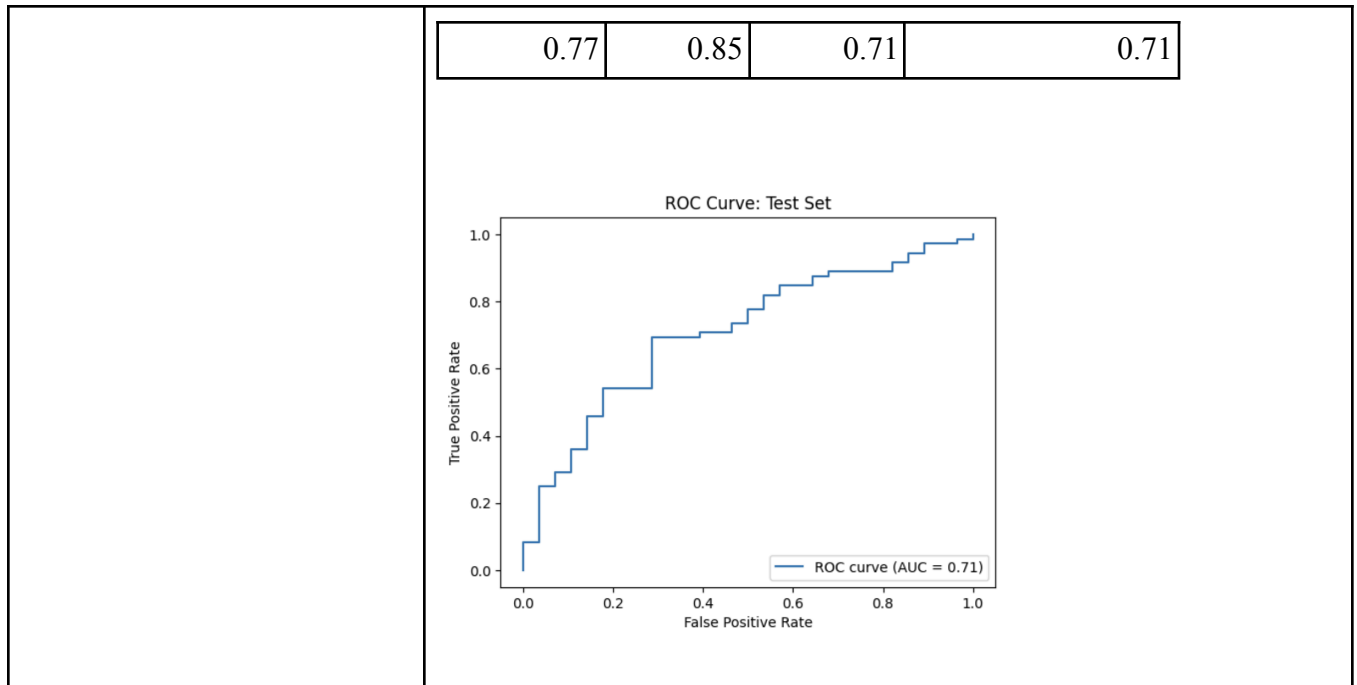
- L2 regularization= 10^{-6} (proportional to regularization strength)
 - Represents a minimal amount of regularization applied

- This suggests that data is not overfitted to the dataset to begin with, which is why applying regularization does not help.
- Learning rate = 10^{-3}
 - Compared with learning rates of 10^{-5} , 10^{-3} , 10^{-1} , 10, 10^3 , 10^5 , 10^7
 - Learning rates too low restrict the model's ability to update parameters within the controlled 1000 iterations of gradient descent. It's updating in steps too little for it to arrive at the optimal set of parameters.
 - Learning rates too high represent taking steps too large during gradient descent. They can cause the model to overshoot the optimal, resulting in a lower validation accuracy.
 - In our case, 10^{-3} appears to be a sweet spot for balancing these two opposing forces.

Comparison of Best Models

Logistic Regression <i>L2 Regularization Strength = 1, Polynomial (degree 3) Feature Transformation</i>	Test Set			
	Precision	Recall	Accuracy	AUC (ROC Curve)
	0.84	0.81	0.75	0.78

	<div><p>ROC Curve (Reg Strength C=1) - Polynomial Test Set</p><p>ROC curve (AUC = 0.78)</p></div>														
<div>SVM</div>	<table><tr><td>Feature Transformation:</td><td></td></tr><tr><td>Best Performance:</td><td>Polynomial Kernel of Degree 3</td></tr><tr><td>Accuracy:</td><td>77%</td></tr><tr><td>Percentage Error:</td><td>23%</td></tr><tr><td>Precision:</td><td>0.78</td></tr><tr><td>Recall:</td><td>0.94</td></tr><tr><td>Hinge Loss:</td><td>0.51</td></tr></table>	Feature Transformation:		Best Performance:	Polynomial Kernel of Degree 3	Accuracy:	77%	Percentage Error:	23%	Precision:	0.78	Recall:	0.94	Hinge Loss:	0.51
Feature Transformation:															
Best Performance:	Polynomial Kernel of Degree 3														
Accuracy:	77%														
Percentage Error:	23%														
Precision:	0.78														
Recall:	0.94														
Hinge Loss:	0.51														
<div>Neural Network</div> <div>10 hidden layers, L2 regularization= 10^-6, Learning rate = 0.001</div>	<table><tr><td colspan="4">Test Set</td></tr><tr><td>Precision</td><td>Recall</td><td>Accuracy</td><td>AUC (ROC Curve)</td></tr></table>	Test Set				Precision	Recall	Accuracy	AUC (ROC Curve)						
Test Set															
Precision	Recall	Accuracy	AUC (ROC Curve)												



Overall, using a support vector machine, with a polynomial kernel of degree 3, produced the highest test set accuracy of 77%, along with the lowest hinge loss of 0.51 among all other SVM models tested. By using a polynomial kernel, the data was able to be mapped to higher dimensions without actually being transformed. This allows for more generalized, simpler decision boundaries to be developed. This leads to a larger margin of error, and a higher likelihood of having points within the margin to be classified correctly.

Additionally, the polynomial's degree was 3, which suggests a generally lower complexity, and more flexible decision boundaries. This allows for better classification, while also capturing complex patterns in new, unseen data as opposed to noise. Therefore, it helps establish a good balance and ensures that the model is not too simple and underfits, but not too complex and overfits. This kind of small transformation was ideal for this model, given its originally high accuracy of 70% on the test set.