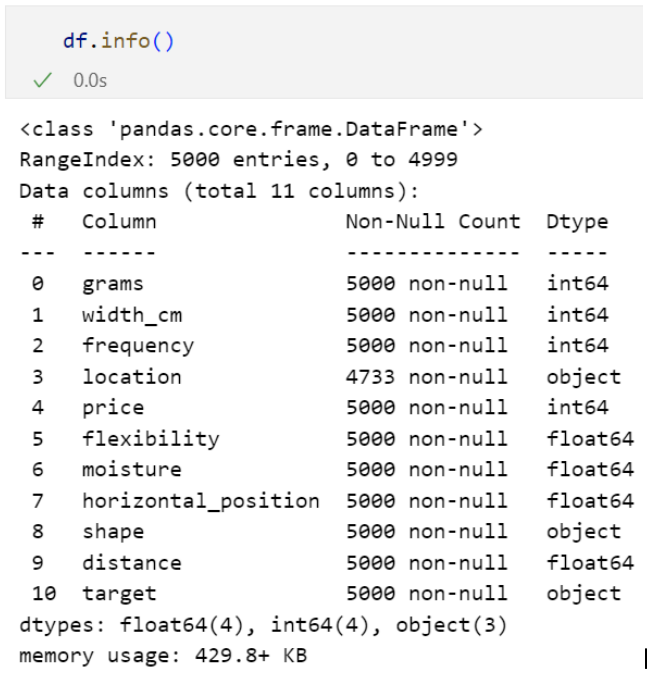
Learning From Data – Individual Repot

Hgwc71

1. Introduction

The report examines the "lfd_2023_group8" dataset using machine learning techniques for training and prediction. It consists of two main sections: the first section details the Support Vector Machine (SVM) training results, while the second section centers on neural network training. Both sections analyze accuracy and training time, showcased via graphical representations.

2. Data Preprocessing



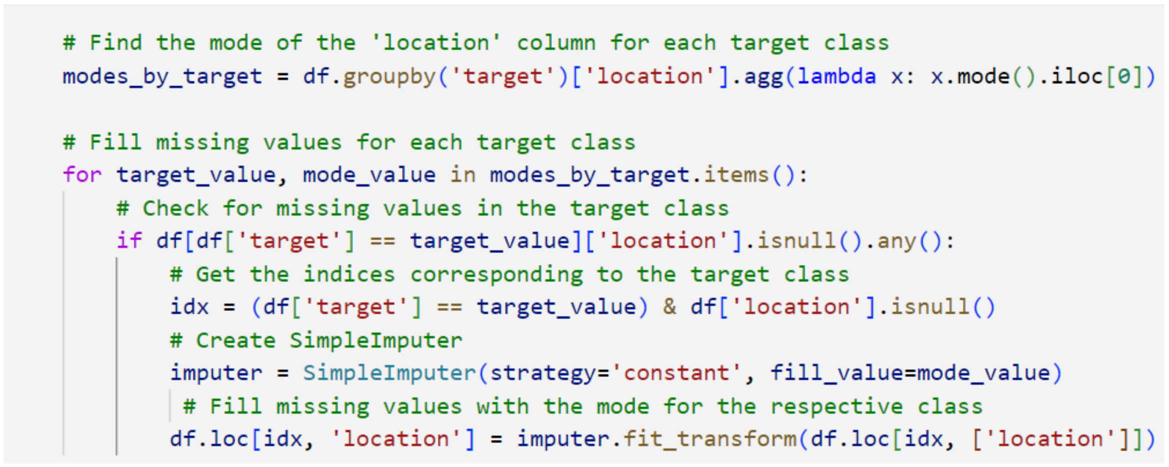
```
df.info()
✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   grams                5000 non-null   int64
 1   width_cm             5000 non-null   int64
 2   frequency            5000 non-null   int64
 3   location             4733 non-null   object
 4   price                5000 non-null   int64
 5   flexibility          5000 non-null   float64
 6   moisture             5000 non-null   float64
 7   horizontal_position  5000 non-null   float64
 8   shape                5000 non-null   object
 9   distance             5000 non-null   float64
 10  target               5000 non-null   object
dtypes: float64(4), int64(4), object(3)
memory usage: 429.8+ KB
```

Figure 1 - the information of dataset

Referring to Figure 1, it's evident that the “location” column contains some missing data denoted by 267 units. Additionally, the varying types of values in each column suggest that the “location” and “shape” columns consist of categorical data.

2.1. Handling Missing Values

When handling missing data, a common practice suggests that if less than 5% of observations are missing, deletion of those values is acceptable. However, exceeding this threshold (as seen in this dataset with 5.34% missing data amounting to 267 instances) can reduce sample size and increase standard errors in parameter estimation (Lebovic, 2015). To mitigate potential issues in model predictions, Sklearn offers imputation tools for filling missing values, a method applied in this dataset.



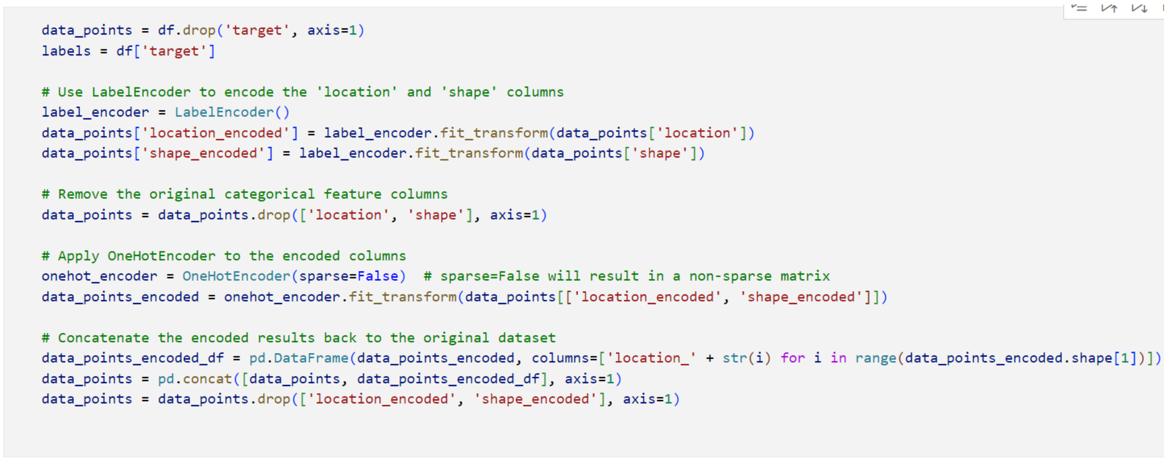
```python
# Find the mode of the 'location' column for each target class
modes_by_target = df.groupby('target')['location'].agg(lambda x: x.mode().iloc[0])

# Fill missing values for each target class
for target_value, mode_value in modes_by_target.items():
    # Check for missing values in the target class
    if df[df['target'] == target_value]['location'].isnull().any():
        # Get the indices corresponding to the target class
        idx = (df['target'] == target_value) & df['location'].isnull()
        # Create SimpleImputer
        imputer = SimpleImputer(strategy='constant', fill_value=mode_value)
         # Fill missing values with the mode for the respective class
        df.loc[idx, 'location'] = imputer.fit_transform(df.loc[idx, ['location']])
```

Figure 2 - the code of imputer for missing data in “location” column

According to Figure 2, the SimpleImputer function is employed to impute the mode as the missing values pertain to categorical data.

2.2. Handling Categorical Values

Figure 1 displays that the 'location' and 'shape' columns contain categorical data. SVM adeptly handles categorical target variables, making it suitable for binary and multiclass classification without additional transformations. Therefore, the focus lies solely on converting the 'location' and 'shape' data into numerical formats.



```python
data_points = df.drop('target', axis=1)
labels = df['target']

# Use LabelEncoder to encode the 'location' and 'shape' columns
label_encoder = LabelEncoder()
data_points['location_encoded'] = label_encoder.fit_transform(data_points['location'])
data_points['shape_encoded'] = label_encoder.fit_transform(data_points['shape'])

# Remove the original categorical feature columns
data_points = data_points.drop(['location', 'shape'], axis=1)

# Apply OneHotEncoder to the encoded columns
onehot_encoder = OneHotEncoder(sparse=False)  # sparse=False will result in a non-sparse matrix
data_points_encoded = onehot_encoder.fit_transform(data_points[['location_encoded', 'shape_encoded']])

# Concatenate the encoded results back to the original dataset
data_points_encoded_df = pd.DataFrame(data_points_encoded, columns=['location_' + str(i) for i in range(data_points_encoded.shape[1])])
data_points = pd.concat([data_points, data_points_encoded_df], axis=1)
data_points = data_points.drop(['location_encoded', 'shape_encoded'], axis=1)
```

Figure 3 - the code to address categorical values in “location” and “shape” columns

3. SVM Model Training Results and Analysis

3.1. Data Scaling

The goal of SVM is to find the maximum-margin hyperplane to separate different classes of data. If one feature has a much larger range compared to the other features, the model may tend to overly emphasize this larger-ranged feature, leading to unfair weighting during the process of maximizing the margin. The following comparison will try to use the linear kernel and preset the C value to 1 first.

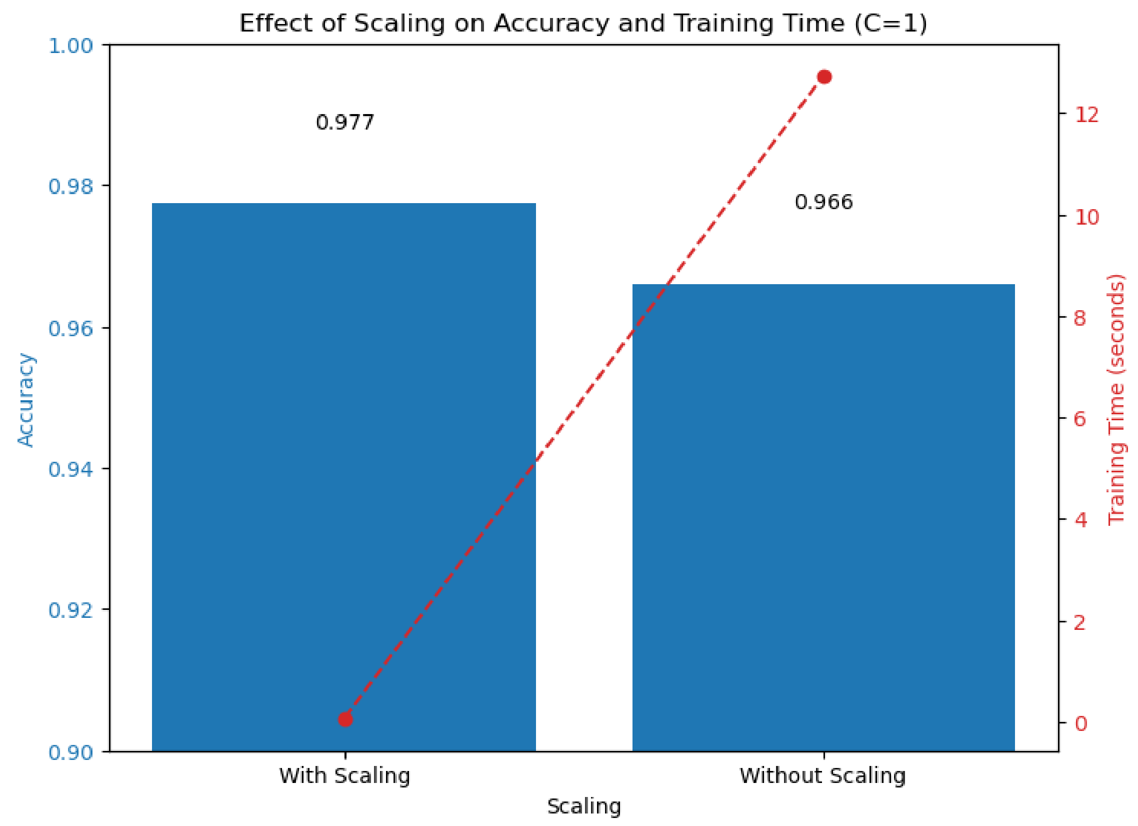


Figure 4 - the effect of scaling on accuracy and training time

Figure 4 illustrates that scaling or omitting feature scaling has a marginal impact on accuracy but significantly affects training time. This disparity arises from the fact that optimization algorithms tend to converge more readily with feature scaling. By narrowing the range of feature values, scaling balances the optimization problem, potentially leading to quicker convergence and reduced training time.

3.2. Hyperparameters Tuning

SVM hyperparameters encompass kernel types (linear, polynomial, RBF), C-value, and Gamma-value. These will be adjusted across different kernels to determine the optimal model.

3.2.1 Linear Kernel

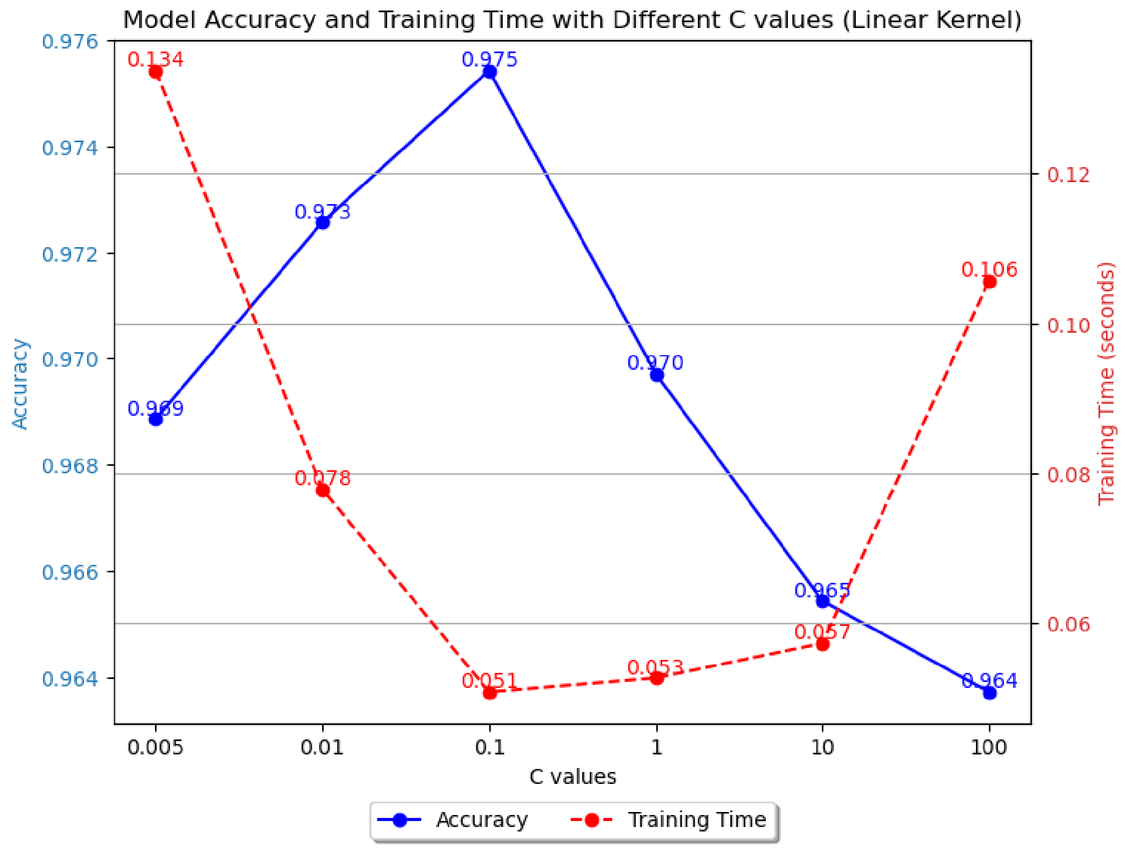
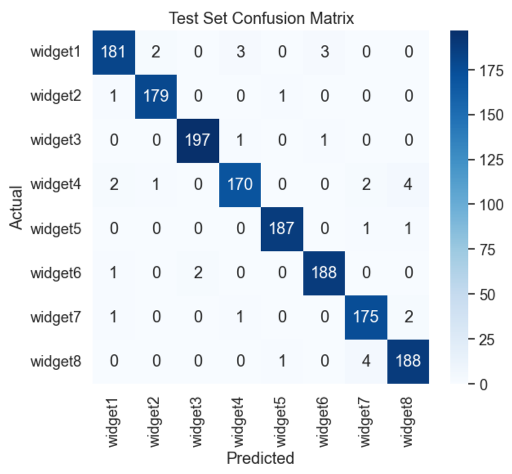
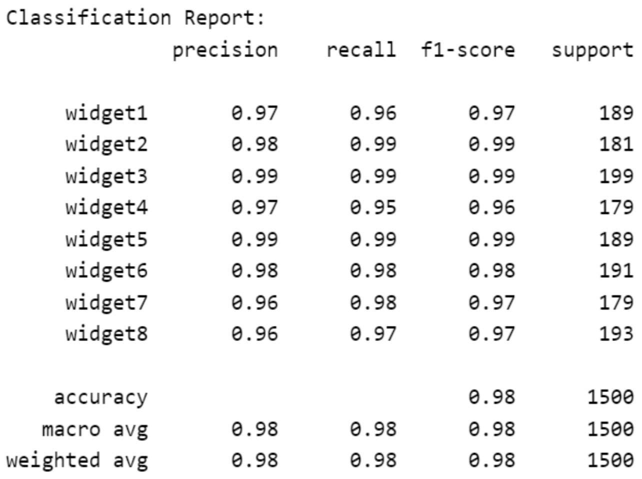


Figure 5 - the effect of different C values on accuracy and training time in linear kernel

Figure 5 demonstrates how altering C values in a linear kernel affects training time. Extreme values, either large or small, prolong training. Larger C values create strict, complex boundaries, while smaller ones struggle with intricate datasets. Setting C to 0.1 notably achieves the highest accuracy and shortest training time. Test set predictions reach an impressive 98% accuracy with C set at 0.1.



	widget1	widget2	widget3	widget4	widget5	widget6	widget7	widget8
widget1	181	2	0	3	0	3	0	0
widget2	1	179	0	0	1	0	0	0
widget3	0	0	197	1	0	1	0	0
widget4	2	1	0	170	0	0	2	4
widget5	0	0	0	0	187	0	1	1
widget6	1	0	2	0	0	188	0	0
widget7	1	0	0	1	0	0	175	2
widget8	0	0	0	0	1	0	4	188



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

     widget1       0.97      0.96      0.97       189
     widget2       0.98      0.99      0.99       181
     widget3       0.99      0.99      0.99       199
     widget4       0.97      0.95      0.96       179
     widget5       0.99      0.99      0.99       189
     widget6       0.98      0.98      0.98       191
     widget7       0.96      0.98      0.97       179
     widget8       0.96      0.97      0.97       193

    accuracy                           0.98      1500
   macro avg       0.98      0.98      0.98      1500
weighted avg       0.98      0.98      0.98      1500
```

Figure 6 - Test set confusion matrix and classification report in a linear kernel (C=0.1)

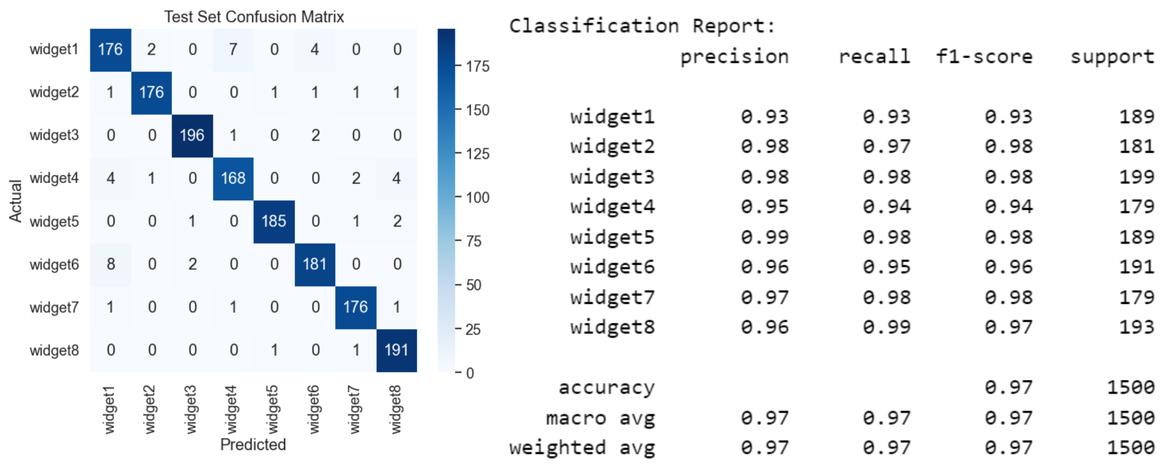
3.2.2 Polynomial Kernel

Given the complexity in visualizing the joint impact of Gamma and C values on accuracy and training time, the emphasis will be on determining the optimal hyperparameter combination. This will be achieved using the Gridsearch method for both polynomial and RBF kernels.

Through Gridsearch, it helps to find the best combination of Gamma and C values for polynomial kernel, which Gamma is 0.1 and C is 0.1.

The for-loop analysis showed consistently low accuracy when pairing extremely small Gamma values with any C value. This was due to the overly simplistic kernel function resulting from excessively small Gamma, which couldn't capture intricate data relationships. Despite increased error penalties, the inadequacy persisted, hindering the model's ability to discern patterns within the data.

Figure 7 illustrates the generalization of the polynomial model. Except for widgets 1 and 4, the prediction accuracy for the remaining widgets reaches almost 97%



	widget1	widget2	widget3	widget4	widget5	widget6	widget7	widget8
widget1	176	2	0	7	0	4	0	0
widget2	1	176	0	0	1	1	1	1
widget3	0	0	196	1	0	2	0	0
widget4	4	1	0	168	0	0	2	4
widget5	0	0	1	0	185	0	1	2
widget6	8	0	2	0	0	181	0	0
widget7	1	0	0	1	0	0	176	1
widget8	0	0	0	0	1	0	1	191

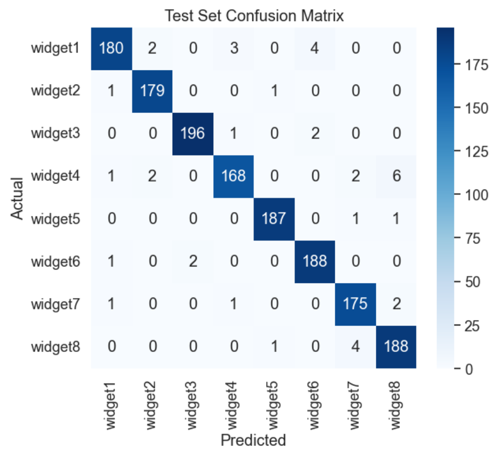
```
Classification Report:
              precision    recall  f1-score   support

     widget1       0.93      0.93      0.93       189
     widget2       0.98      0.97      0.98       181
     widget3       0.98      0.98      0.98       199
     widget4       0.95      0.94      0.94       179
     widget5       0.99      0.98      0.98       189
     widget6       0.96      0.95      0.96       191
     widget7       0.97      0.98      0.98       179
     widget8       0.96      0.99      0.97       193

    accuracy                           0.97      1500
   macro avg       0.97      0.97      0.97      1500
weighted avg       0.97      0.97      0.97      1500
```

Figure 7 - Test set confusion matrix and classification report in a polynomial kernel (C=0.1, Gamma=0.1)

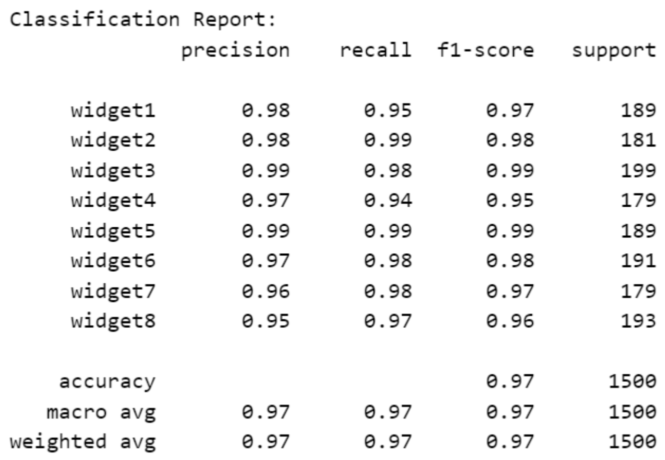
3.2.3 RBF Kernel

Gridsearch aids in identifying the optimal combination of Gamma and C values for the RBF kernel, pinpointing Gamma at 0.005 and C at 10.

While examining various combinations, we noticed that training time didn't exhibit a clear linear relationship with adjustments in both hyperparameters. Regarding accuracy, lower C values showed no significant improvement in precision even with increased Gamma. However, a rise in C to 0.1 notably boosted accuracy with higher Gamma values. Yet, for larger C values, the impact of adjusting Gamma on accuracy gradually diminished.



	widget1	widget2	widget3	widget4	widget5	widget6	widget7	widget8
widget1	180	2	0	3	0	4	0	0
widget2	1	179	0	0	1	0	0	0
widget3	0	0	196	1	0	2	0	0
widget4	1	2	0	168	0	0	2	6
widget5	0	0	0	0	187	0	1	1
widget6	1	0	2	0	0	188	0	0
widget7	1	0	0	1	0	0	175	2
widget8	0	0	0	0	1	0	4	188



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

     widget1       0.98      0.95      0.97       189
     widget2       0.98      0.99      0.98       181
     widget3       0.99      0.98      0.99       199
     widget4       0.97      0.94      0.95       179
     widget5       0.99      0.99      0.99       189
     widget6       0.97      0.98      0.98       191
     widget7       0.96      0.98      0.97       179
     widget8       0.95      0.97      0.96       193

    accuracy                           0.97      1500
   macro avg       0.97      0.97      0.97      1500
weighted avg       0.97      0.97      0.97      1500
```

Figure 8 - Test set confusion matrix and classification report in a rbf kernel (C=10, Gamma=0.005)

By observing Figure 8, it's evident that the RBF kernel demonstrates a high level of predictive accuracy on this test set, achieving an overall accuracy of 97%.

4. Neural Network Training Results and Analysis

In the training of neural networks, there exist several adjustable hyperparameters, including:

- Learning rate

  The learning rate will directly affect the convergence state of the model. To balance the accuracy and training time, the learning rate will be set to start from 0.001 and increase to 0.005, 0.01,0.015.

- Batch size

  Smith et al. (2018) proposed that keeping a consistent learning rate while scaling up the batch size might enhance test set accuracy. This phenomenon, observed in certain experiments and datasets, indicates that larger batch sizes could enhance the model's generalization, leading to improved performance on test sets. Hence, in subsequent training, we'll explore the impact on accuracy and training time by increasing the batch size while keeping the learning rate constant.

- Epochs

  To enhance the model's generalization capability and reduce training loss, the value of epochs will initially be set to 100 to explore its variations.

- Number of layers and neurons

  Regarding the number of hidden layers and neurons, Jeff Heaton (2017) mentioned, "Problems that require more than two hidden layers were rare prior to deep learning. Two or fewer layers will often suffice with simple data sets. However, with complex datasets involving time-series or computer vision, additional layers can be helpful." Thus, with respect to the number of hidden layers, training will be conducted using 1 and 2 layers. As for the number of neurons, reference will be made to the guidelines provided by Heaton (2017):

A. The number of hidden neurons should be between the size of the input layer and the size of the output layer.
B. The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
C. The number of hidden neurons should be less than twice the size of the input layer. (Heaton, 2017)

- Choice of activation function

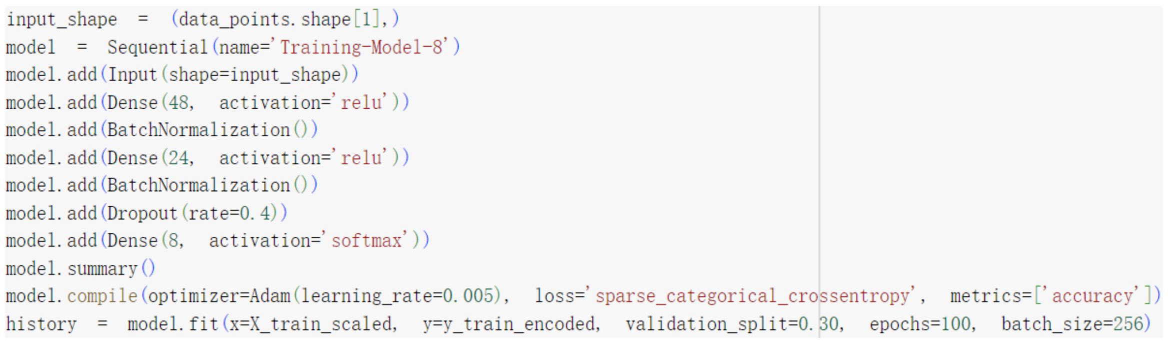
For a multi-class classification problem with 10 features and 8 classes, will use the 'ReLU'. This activation function effectively mitigates the issue of gradient vanishing due to the depth of the network layers, while also boosting computational efficiency.

- Choice of optimization algorithm

The optimization algorithm will choose Adam, because as go-to optimizer, Adam is better than other adaptive learning rate algorithms due to its faster convergence and robustness across problems. (Agarwal, 2023)

- Dropout ratio

The dropout ratio is commonly set between 0.2 to 0.5 based on general empirical experience and common practices.



```
input_shape = (data_points.shape[1],)
model = Sequential(name='Training-Model-8')
model.add(Input(shape=input_shape))
model.add(Dense(48, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(24, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(rate=0.4))
model.add(Dense(8, activation='softmax'))
model.summary()
model.compile(optimizer=Adam(learning_rate=0.005), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x=X_train_scaled, y=y_train_encoded, validation_split=0.30, epochs=100, batch_size=256)
```

Figure 9 - Neural network structure

Figure 9 displays the optimized hyperparameters for the neural network model. The process involved choosing the learning rate with the shortest training time initially. Subsequently, prioritization was based on achieving the highest training accuracy and lowest loss. The final selected hyperparameters are outlined below:

- Learning rate: 0.005
- Batch size: 256
- Epochs: 100
- Number of layers and neurons: 2 dense layers, first layer includes 48 neurons and second layer includes 24 neurons.
- Dropout ratio: 0.4

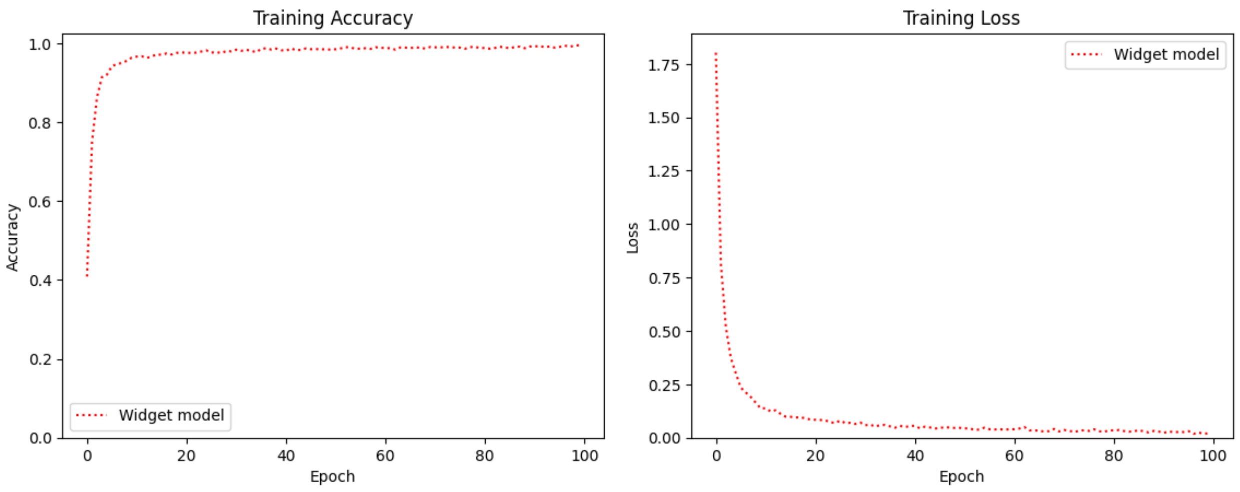
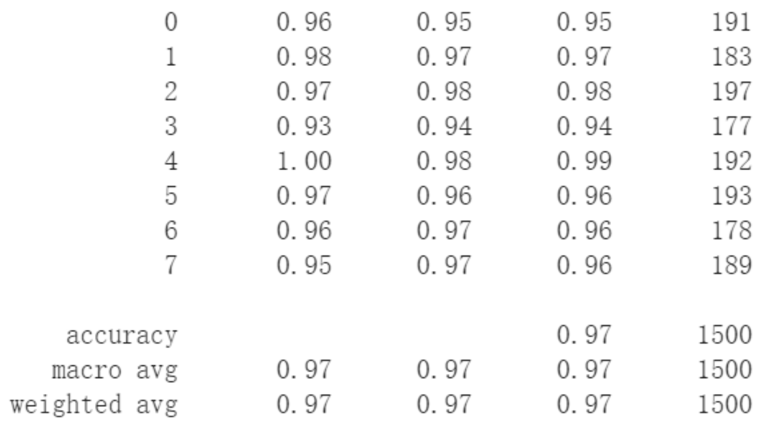


Figure 10 - The training accuracy and training loss of optimal model



```
           0       0.96      0.95      0.95       191
           1       0.98      0.97      0.97       183
           2       0.97      0.98      0.98       197
           3       0.93      0.94      0.94       177
           4       1.00      0.98      0.99       192
           5       0.97      0.96      0.96       193
           6       0.96      0.97      0.96       178
           7       0.95      0.97      0.96       189

    accuracy                           0.97      1500
   macro avg       0.97      0.97      0.97      1500
weighted avg       0.97      0.97      0.97      1500
```

Figure 11 - Test set classification report in neural network

The model's accuracy improves, and loss decreases progressively with longer epochs, signifying its gradual optimization. Figure 11's depiction of high-test set accuracy validates that the model avoids overfitting.

5. Comparison and Summary

Type of model	Training accuracy	Training time	Testing accuracy
SVM - linear	0.975	0.051s	0.98
SVM - Polynomial	0.965	0.159s	0.97

SVM - rbf	0.976	0.233s	0.97
Neural network	0.995	22.50s	0.97

In summary, the models show similar performance on the test set, ranging from around 0.97 to 0.98 accuracy. The neural network achieved the highest training accuracy but required more time. SVM models with different kernels had relatively similar test accuracies, with the linear kernel having the shortest training time. Considering the trade-off between training time and test accuracy, opting for the linear SVM model strikes a balance between these factors.

References

[1] Zvornicanin, E. (2023, March 16). *Relation Between Learning Rate and Batch Size*. Baeldung. https://www.baeldung.com/cs/learning-rate-batch-size

[2] Heaton, J. (2017, June 1). *The Number of Hidden Layers*. Heaton Research. https://www.heatonresearch.com/2017/06/01/hidden-layers.html

[3] Yastremsky, D. (2021, March 4). *Exploit Your Hyperparameters: Batch Size and Learning Rate as Regularization*. Towards Data Science. https://towardsdatascience.com/exploit-your-hyperparameters-batch-size-and-learning-rate-as-regularization-9094c1c99b55

[4] Agarwal, R. (2023, September 13). *Complete Guide to the Adam Optimization Algorithm*. Builtin. https://builtin.com/machine-learning/adam-optimization

[5] Brownlee, J. (2019, August 6). *How to Configure the Learning Rate When Training Deep Learning Neural Networks*. Machine Learning Mastery. https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

[6] Gerald, L. (2015, June). *Bridging the Data Gap: How to Deal with Missing Data in Observational Studies*. The HUB. https://www.hubresearch.ca/bridging-the-data-gap-how-to-deal-with-missing-data-in-observational-studies/