



Data Science Challenge (DSC) Problem Development

August 7th, 2024

Kayla Pascua



Data Science Challenge (DSC)

What is it?

- Lab developed dataset students run machine learning and data science techniques on
- 2 week program

Who participates?

- Selected undergraduate/graduate students
- DSSI interns

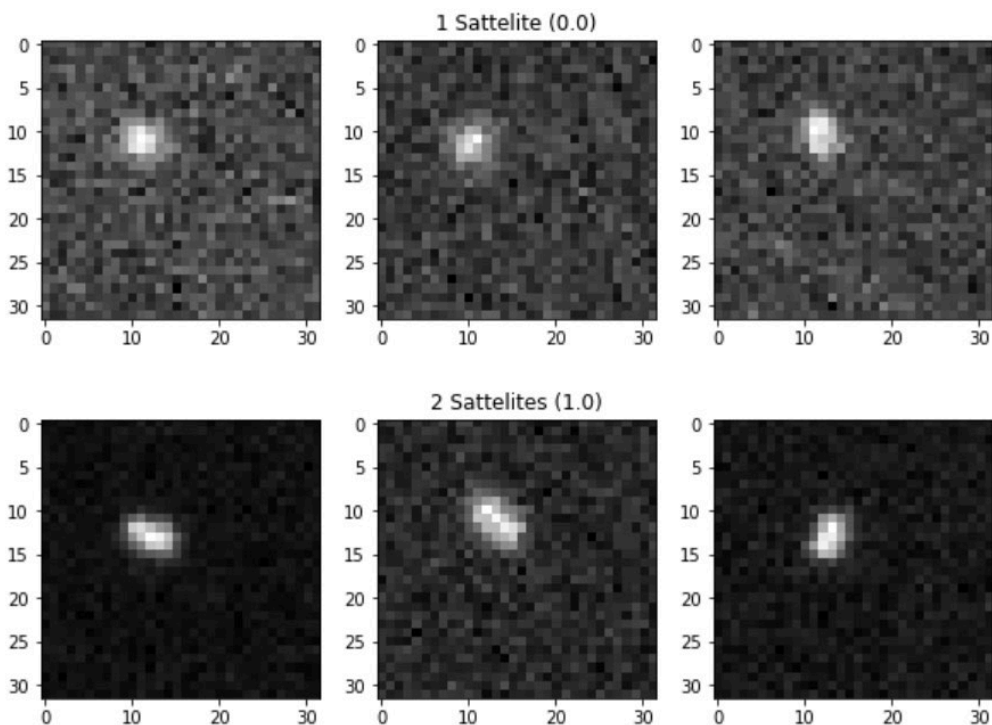
This year's challenge:

- Working with electrocardiogram data
- Classifying types of heartbeats
- Reconstruction using neural networks



2025 Challenge Problem

- Ground-based simulated space domain awareness data
- Tasks:
 - Classifying 1 or 2 satellites
 - Distance between 2 satellites
 - Finding the magnitude difference between 2 satellites



```

jupyter DSC 2023 Last Checkpoint: 5 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (pykernel) O
In [385]: # Dropping 'label' column
data = df.drop(columns="label")
# Turning data into an array of values
dataArr = np.array(data, dtype=np.float32)

In [386]: # Normalizing data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(dataArr)
imgArr = np.zeros((len(scaled_data), 32, 32), dtype=np.float32)
for i in range(len(scaled_data)):
    imgArr[i] = np.reshape(scaled_data[i], (32, 32))

In [387]: from sklearn.model_selection import train_test_split
X = imgArr
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=30)

In [459]: from keras.callbacks import EarlyStopping
from tensorflow.keras.metrics import Recall, Precision
# es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[accuracy, Recall(), Precision()])
history = model.fit(X_train, y_train, batch_size=100, epochs=50, verbose=1, validation_data=(X_test, y_test))

Epoch 1/50 3s 32ms/step - accuracy: 0.5465 - loss: 0.7627 - precision_24: 0.5495 - recall_24: 0.487
60/60 3 - val_accuracy: 0.5734 - val_loss: 0.6757 - val_precision_24: 0.6844 - val_recall_24: 0.3837
Epoch 2/50 60/60 1s 18ms/step - accuracy: 0.6060 - loss: 0.6997 - precision_24: 0.6183 - recall_24: 0.530
0 - val_accuracy: 0.5805 - val_loss: 0.6676 - val_precision_24: 0.7941 - val_recall_24: 0.2396
Epoch 3/50 60/60 1s 18ms/step - accuracy: 0.6379 - loss: 0.6666 - precision_24: 0.6563 - recall_24: 0.526
7 - val_accuracy: 0.5236 - val_loss: 0.6903 - val_precision_24: 0.9241 - val_recall_24: 0.8720
Epoch 4/50 60/60 1s 18ms/step - accuracy: 0.6383 - loss: 0.6596 - precision_24: 0.6496 - recall_24: 0.579
6 - val_accuracy: 0.5739 - val_loss: 0.6647 - val_precision_24: 0.9034 - val_recall_24: 0.1844
Epoch 5/50 60/60 1s 18ms/step - accuracy: 0.6645 - loss: 0.6333 - precision_24: 0.6773 - recall_24: 0.585
7 - val_accuracy: 0.5893 - val_loss: 0.6595 - val_precision_24: 0.9123 - val_recall_24: 0.2155
Epoch 6/50 60/60 1s 18ms/step - accuracy: 0.6690 - loss: 0.6050 - precision_24: 0.6914 - recall_24: 0.590
0 - val_accuracy: 0.5777 - val_loss: 0.6731 - val_precision_24: 0.9589 - val_recall_24: 0.1815
    
```

	label	pixel_0	pixel_1	pixel_2	pixel_3	pixel_4	pixel_5	pixel_6	pixel_7	pixel_8	...	pixel_1014	pixel_1015	pixel_1016
0	0.0	101.663208	101.263336	92.915863	120.369873	114.871735	76.957855	83.237640	106.683411	91.402435	...	55.439087	108.598663	91.091034
1	0.0	8.028961	9.625214	28.555206	41.477539	29.168304	19.793213	9.517181	-8.416779	3.215607	...	23.384827	8.169556	25.309662
2	0.0	14.992676	48.896774	21.896484	50.559189	7.921494	31.539688	14.531250	30.757767	42.491211	...	7.537674	18.652588	30.966095
3	0.0	2.685196	7.460678	4.135025	-0.644028	22.422089	24.757065	-3.759689	25.751801	52.419250	...	33.456985	2.220596	14.763306
4	0.0	13.167809	6.210907	14.006561	2.071999	20.662659	27.482124	13.024620	20.394508	21.195366	...	16.936974	-6.974487	24.083168
...
9935	1.0	28.073456	6.037537	53.251770	38.230133	50.442474	49.583099	40.899811	36.529449	-11.407104	...	69.461975	15.480469	15.737732
9936	1.0	19.476059	7.419373	25.825912	-0.350510	36.773300	31.415192	8.433899	-10.451599	11.901138	...	8.683578	28.986221	4.287933
9937	1.0	7.257568	12.798042	46.397675	11.509476	7.482903	8.362656	15.834869	11.505180	19.770462	...	13.685669	8.806023	28.292709
9938	1.0	29.861664	13.363983	22.918152	16.295456	19.118454	30.038254	26.174240	26.320053	-1.169106	...	40.680313	25.710754	17.670853
9939	1.0	81.447693	56.053528	97.905930	57.954575	88.921417	74.050003	66.869675	75.146255	52.953522	...	74.379257	88.548676	61.582718

9940 rows x 1025 columns

Tutorials

- Summer students come in with various levels of knowledge
- Jupyter notebooks to learn basic ML/DS techniques

Confusion Matrix

- True Positive (TP) – Predicted value matches actual (positive)
- False Positive (FP) – Value predicted as positive when actual is negative
- True Negative (TN) – Predicted value matches actual (negative)
- False Negative (FN) – Value predicted as false when actual is positive

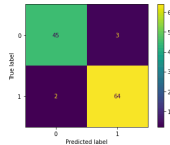
```
n [16]: # Creating
TP = ((y_test == 1) & (y_predict == 1)).sum()
TN = ((y_test == 0) & (y_predict == 0)).sum()
FP = ((y_test == 0) & (y_predict == 1)).sum()
FN = ((y_test == 1) & (y_predict == 0)).sum()

precision = TP / (TP + FP)
recall = TP / (TP + FN)
fpr = FP / (FP + TN)

print(f"TP = {TP}, TN = {TN}, FP = {FP}, FN = {FN}")
print(f"Precision = {precision:.2f}, Recall = {recall:.2f}, False Positive Rate = {fpr:.2f}")

TP = 64, TN = 45, FP = 3, FN = 2
Precision = 0.96, Recall = 0.97, False Positive Rate = 0.06
```

```
n [17]: # Using a sklearn
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_predict)
ConfusionMatrixDisplay(confusion_matrix=cm).plot();
```



Classification Accuracy

- Determines how accurate the model's prediction is to the correct output
- Calculate: true values added / all values added

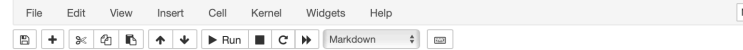
Misclassification Rate (Error Rate)

- How often the model makes incorrect predictions
- Calculate: false values added / all values added

Precision

- TP / (TP + FP)
- Gets percent of true positive predictions

Jupyter Convolutional Neural Network Last Checkpoint: 5 hours ago (autosaved)



Introduction

Also known as an artificial neural networks (ANN), a neural network is made up of layers that consist of nodes, these nodes that interpret the data taken in a certain way.

Weights are used to pick out certain variables within the data to compare other variables against the outputted result. It is the activation function, which will determine its output. If the data has passed the threshold, it will move on to the next layer with

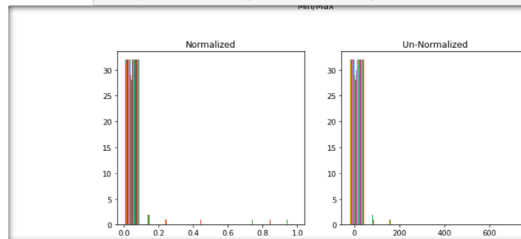
What is a convolutional neural network?

Jupyter Normalization Techniques Last Checkpoint: 06/11/2024 (unsaved changes)



Comparing

```
In [8]: normGraphs(minmaxArr, imgArr, random.randrange(len(imgArr)), "Min/Max")
normGraphs(zscoreArr, imgArr, random.randrange(len(zscoreArr)), "Z-Score")
```



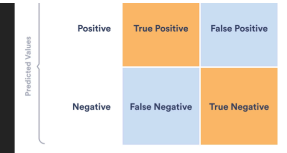
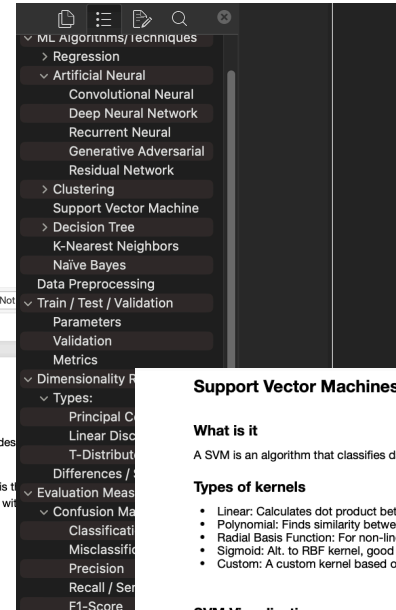
Disadvantages

- Loss of information
 - How information is interpreted
 - Detection of outliers
- May complicate un-normalized data
- Hard to query because it is meant for programmes, not ad hoc queries

ML Techniques that Use Data Normalization

Used for algorithms that use distance measurements

- Logistic Regression
- Artificial Neural Networks (ANN)
- Neural Networks (multi-layered perceptron)



Classification Accuracy

- Determines how accurate the model's prediction is to the correct output
- Calculate: true values added / all values added

Misclassification Rate (Error Rate)

- How often the model makes incorrect predictions
- Calculate: false values added / all values added

Precision

Gets percent of true positive predictions

Support Vector Machines (SVMs)

What is it

A SVM is an algorithm that classifies data by fitting a line to maximize the distance between each class

Types of kernels

- Linear: Calculates dot product between vectors
- Polynomial: Finds similarity between 2 vectors in respect to polynomial of original variables
- Radial Basis Function: For non-linear decision boundaries, maps data to infinite dimensional spaces
- Sigmoid: Alt. to RBF kernel, good for neural networks and non-linear classifiers
- Custom: A custom kernel based on given domain information

SVM Visualization

```
n [47]: from sklearn.svm import SVC
X = iris.data[:, :2]
clf_linear = SVC(kernel='linear')
clf_linear.fit(X, y)
```

```
out[47]: SVC(kernel='linear')
```

```
n [48]: h = 0.02 # step size in the mesh
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                    np.arange(y_min, y_max, h))
```

```
n [49]: linear = clf_linear.predict(np.c_[xx.ravel(), yy.ravel()])
linear = Z_linear.reshape(xx.shape)
plt.contourf(xx, yy, Z_linear, cmap=plt.cm.Paired, alpha=0.6)
scatter = plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
handles, labels = scatter.legend_elements()
plt.legend(handles = handles, labels = ["Setosa", "Versicolour", "Virginica"], title="CL")
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.show()
```

