Lazy Learning

Jetwyn Wilson

What is Lazy Learning?

- Machine learning approach where generalization of training data is delayed until a query is made
- Uses similarities using distance to predict
- Able to adapt quickly to new and changing data without training
- Real world applications: Medical diagnosis and anomaly detection
- Useful for high-dimensional spaces and nonlinear tasks

Literature Review

Bayrak (2022)

Real World Applications

- Uses lazy learning methods to in enhance fraud detection
- Uses hybrid lazy learning to adapt to real world task that are high in dimension

J.Liu (2016)

Introducing K-NN

- Uses K-NN to select neighbors based on data distribution to improve accuracy and flexibility
- Embedding techniques helps scalability and feature representation

Y. Liu (2017)

Medical Setting

- Lazy learning methods are great because retraining is not necessary so it can constantly adapt to healthcare when data changes

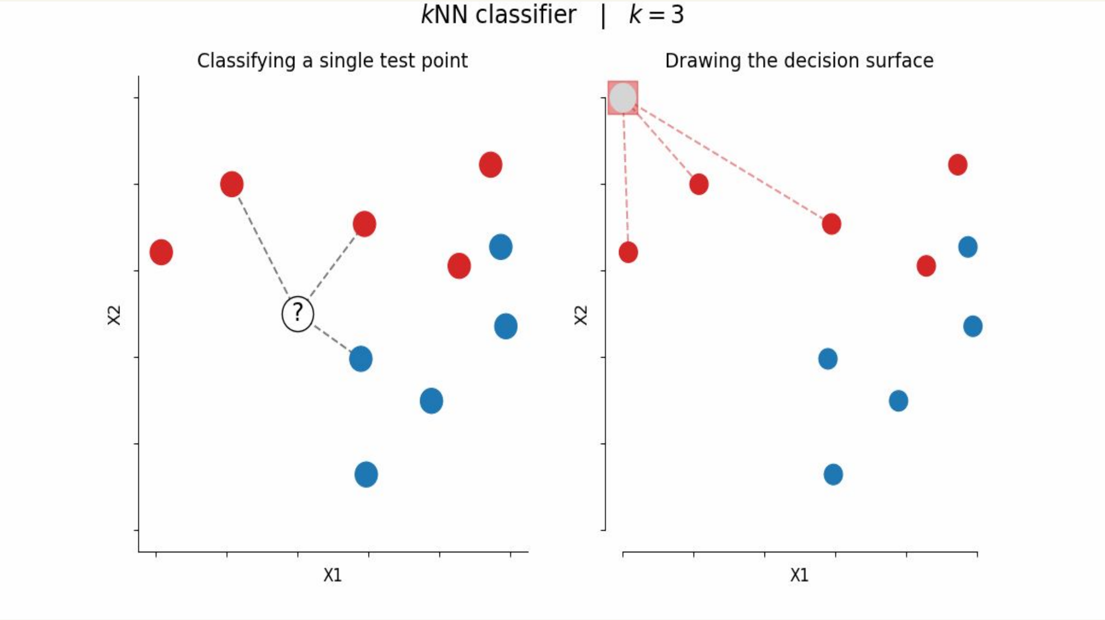
S. Zhou (2020)

Impact of K-NN

- Study how K-NN captures local pattern when feature and labels relationship is strong

K-NN

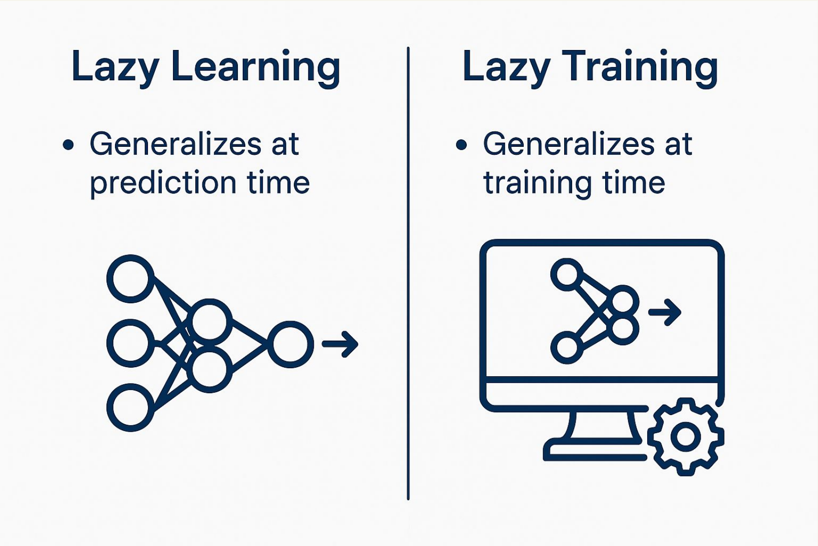
- The lazy learning method that is most used
- Computational Cost is low
- Thrive in an environment where resources are low



Difference between Lazy Learning vs Training

Lazy learning methods has a delay in generalizing until told so. They store the training data to use its inference time to give predictions.

Lazy training still has a training phase but the goal is to minimize it



Comparing with Eagar Learning Methods

Gradient Descent

- Easy to implement and understand
- Effective in reducing error
- Low memory footprint

Stochastic Gradient Descent

- Fast updates
- Suited for limited memory and computation environments
- Escape local minima due to noise

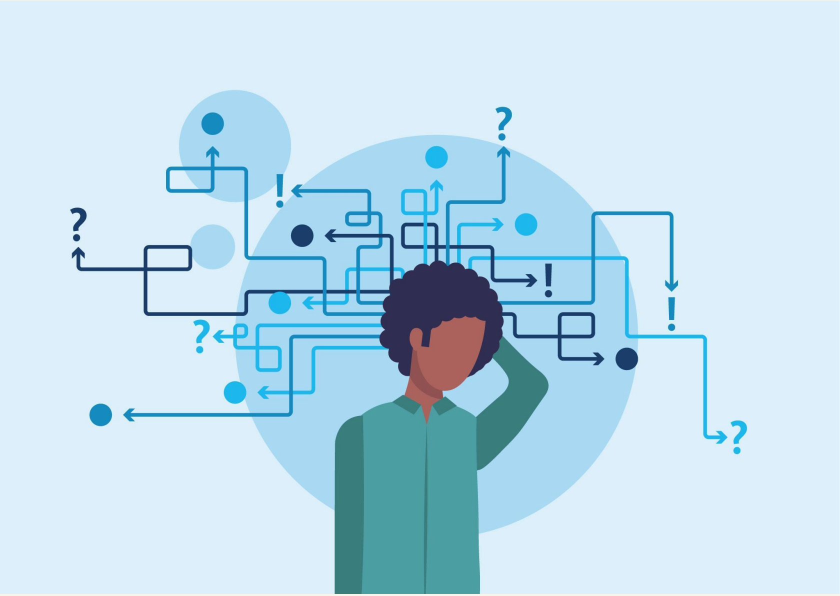
Adam

- Combines momentum and adaptive learning rate
- Fast convergence on complex problems
- Perform well with sparse gradient

Problem Statement

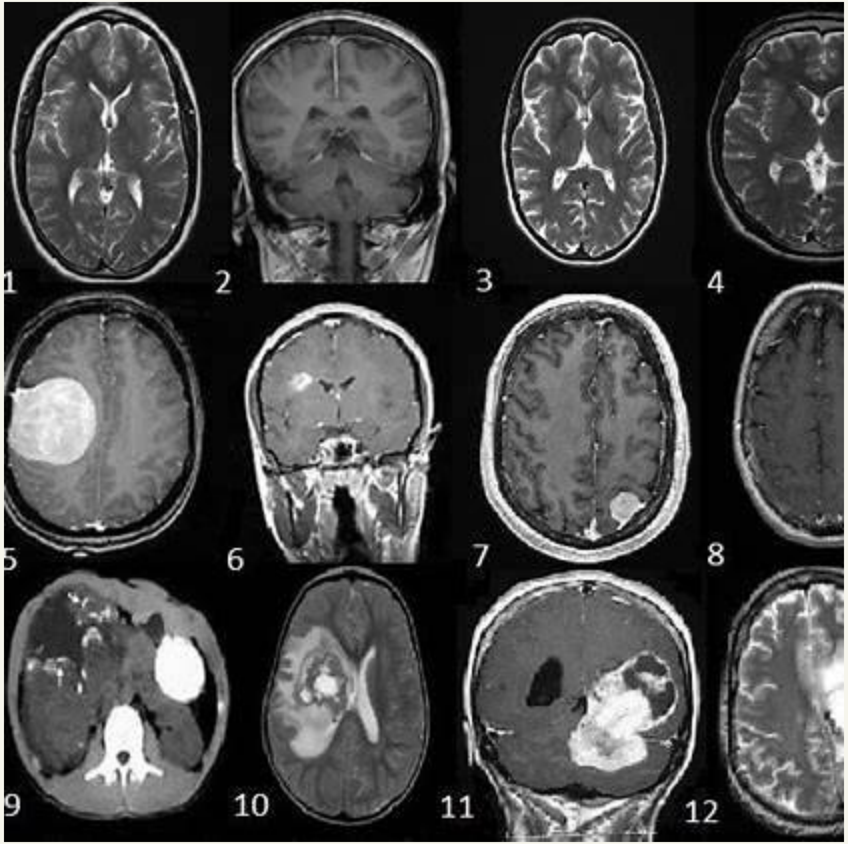
Real Goal?

1. What conditions is lazy learning methods are more favorable?
2. Challenge: Dataset is a classification problem that is nonlinear and has a high dimension
3. Evaluation
   a. Accuracy
   b. Inference time
   c. Memory Usage
   d. Scalability



Introducing Dataset

- Dataset: Letter Recognition
- Total Samples: 20,000
- Features: 16 numerical features to represent the Alphabet
- Target: 26 Uppercase Alphabet
- Dataset: Non-linear, high dimensional classification



How Each was trained?

Training Setup:

- 80% training/ 20% testing split
- Training set reduced to 2,00 samples to stimulate constrained environments

Preprocessing

- Features standardized
- One-hot encoding for class labels (for neural networks)

K-NN

- No training phase
- k = 3, chosen based on best validation accuracy
- Uses Euclidean distance
- Uniform weighting
- Computation deferred until prediction (inference-heavy)

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Initialize and train the KNN classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Predict and evaluate
y_pred_knn = knn.predict(X_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f"K-NN Accuracy: {accuracy_knn:.4f}")
```

Gradient Descent

- Library used SGDClassifier from scikit-learn
- Training setup
    - Mac_iter=1 -to limit training
    - No shuffling od data-one pass batch update
    - Loss function: Logistic regression
    - Learning rate: constant with eta0=0.01
    - No regularization applied

This set stimulates a resource constrained environment to reflect lazy training conditions

```python
# Initialize weights
weights = np.zeros(X_train.shape[1])
learning_rate = 0.01
epochs = 1000

# Gradient Descent loop
for epoch in range(epochs):
    predictions = 1 / (1 + np.exp(-np.dot(X_train, weights)))
    gradient = np.dot(X_train.T, (predictions - y_train)) / y_train.size
    weights -= learning_rate * gradient

# Evaluate
test_preds = 1 / (1 + np.exp(-np.dot(X_test, weights)))
test_preds = test_preds >= 0.5
accuracy_gd = np.mean(test_preds == y_test)
print(f"Gradient Descent Accuracy: {accuracy_gd:.4f}")
```

Stochastic Gradient (SGD)

- Uses SGDClassifier from scikit-learn
- Training setup
  - max_iter=5,
    - Helps convergence because it gives a fair comparison to add fairness
  - Batch size 1
  - Learning rate optimal
  - Loss function: log regression

```python
from sklearn.linear_model import SGDClassifier

# Initialize and train the SGD classifier
sgd = SGDClassifier(loss='log', max_iter=1000, learning_rate='optimal', tol=1e-3)
sgd.fit(X_train, y_train)

# Predict and evaluate
y_pred_sgd = sgd.predict(X_test)
accuracy_sgd = accuracy_score(y_test, y_pred_sgd)
print(f"SGD Accuracy: {accuracy_sgd:.4f}")
```

Adam

Uses Tensorflow
Us a shallow feedforward neural network

- 1 hidden layer with Relu units
- Soft max output for over 26 classes

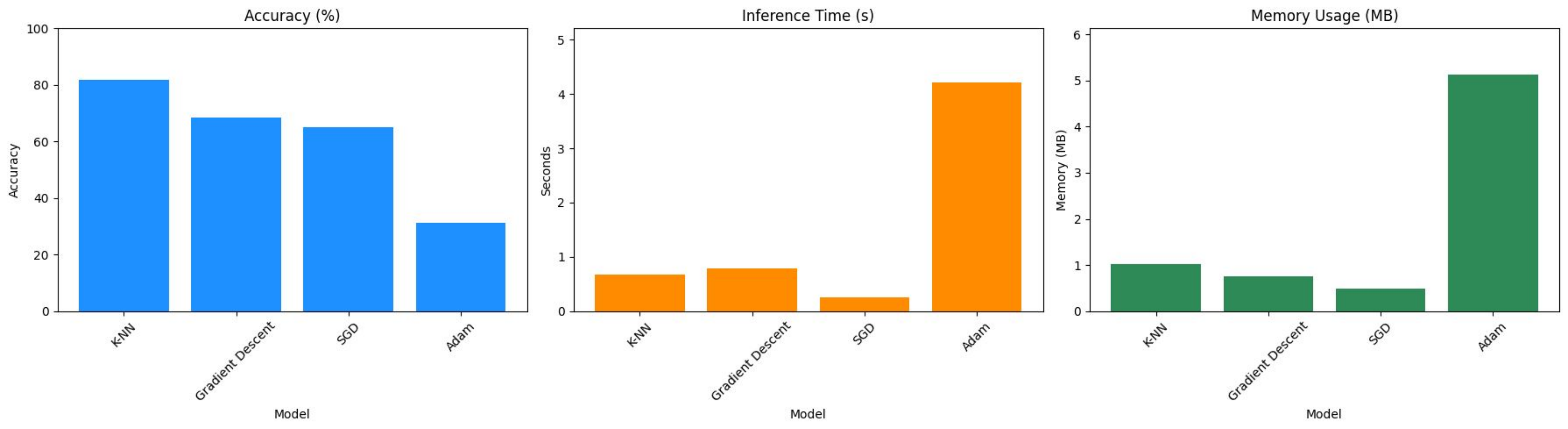
Training setup

- Trained for 1 epoch only
- Batch size 16
- Learning rate
- Weight initialization: Glorot uniform
- No early stopping applied

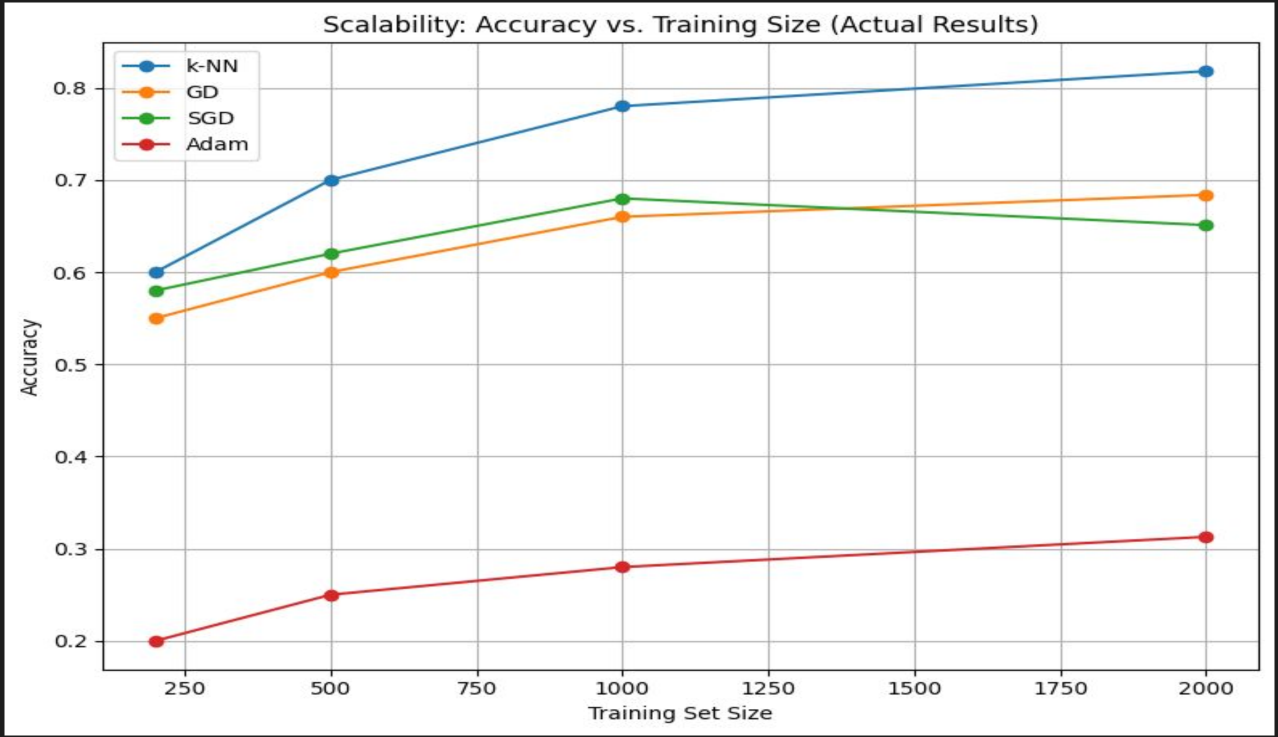
```
class SimpleNN(nn.Module):
    def __init__(self, input_dim):
        super(SimpleNN, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(input_dim, 256),
            nn.ReLU(),
            nn.Dropout(0.4),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.fc(x)
model = SimpleNN(X_train.shape[1])
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.reshape(-1, 1), dtype=torch.float32)

# Training loop
for epoch in range(100):
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# Evaluate (using thresholded predictions)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
with torch.no_grad():
    y_pred_adam = model(X_test_tensor).numpy() >= 0.5
accuracy_adam = np.mean(y_pred_adam.flatten() == y_test)
print(f"Adam Accuracy: {accuracy_adam:.4f}")
```

Results

Comparing Models



Evaluating Results



Real Application Areas

Embedded systems

- Devices like sensors, smartwatches and industrial controllers have limited memory and computation power
- Using K-NN would be well suited because it does not require retraining and adapt quickly to new inputs

Mobile health application

- Benefit from lazy learning because it works well with personalized and constantly changing data
- K-NN makes decision based on individual health patterns without needing to relearn from scratch

Edge Devices

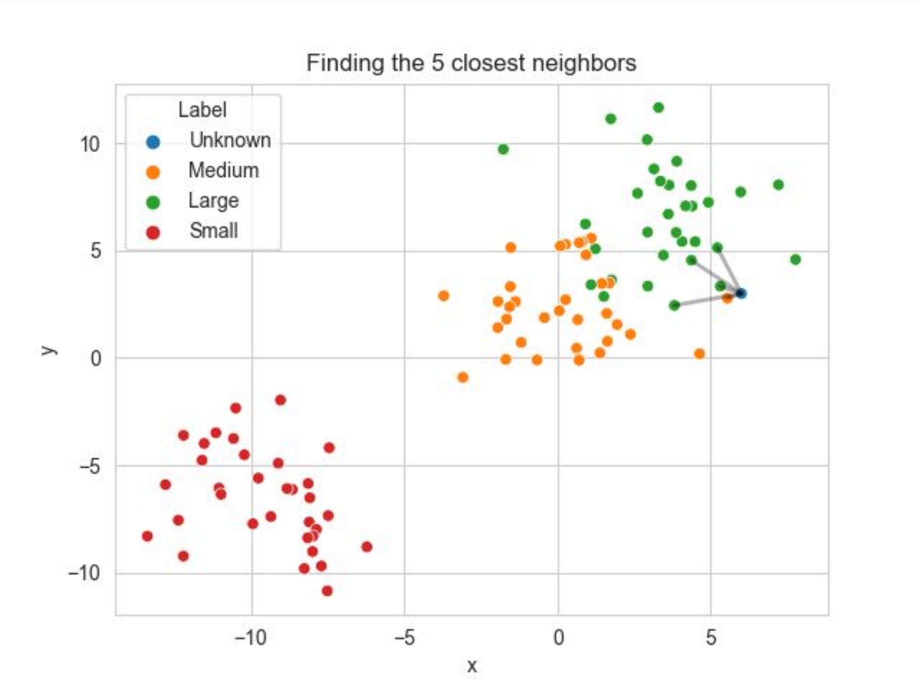
- Devices like surveillance cameras often need fast and low memory predictiction to deliver on the fly inference with minimal computational load
- Helpful for remote environments

Summary

Lay Learning is best used when resources are limited:

Lazy learning is great when you have limited time and grant money because it thrives in

- No training required
- High model adaptability
- High accuracy with low training
- Efficient on device inference



K-NN Algorithm