Week 00a Intermediate Handout: ML Foundations - Hands-On Implementation

Machine Learning for Smarter Innovation

1 Week 00a Intermediate Handout: ML Foundations - Hands-On Implementation

1.1 For Students With: Basic Python, willingness to code

1.2 Objectives

- Implement ML algorithms using scikit-learn
- Understand train/test split and evaluation
- Apply preprocessing and feature engineering
- Build end-to-end ML pipeline

1.3 Setup Environment

```
# Install required packages
pip install scikit-learn pandas numpy matplotlib seaborn

# Import essentials
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

1.4 Exercise 1: Spam Detection (Supervised Classification)

1.4.1 Step 1: Load and Explore Data

```python
# Load SMS spam dataset
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

# For this example, we'll create synthetic spam data
emails = [
    "Free money click here now",
    "Meeting tomorrow at 3pm",
    "Congratulations you won lottery",
    "Can you review my document",
    "Claim your prize urgent",
    "Lunch plans for Friday"
]
labels = [1, 0, 1, 0, 1, 0]  # 1=spam, 0=not spam

# Check class balance
print(f"Spam: {sum(labels)}, Not spam: {len(labels)-sum(labels)}")
```

1.4.2 Step 2: Feature Engineering

```python
# Convert text to numbers using TF-IDF
vectorizer = TfidfVectorizer(max_features=100, stop_words='english')
X = vectorizer.fit_transform(emails)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, labels, test_size=0.3, random_state=42, stratify=labels
)

print(f"Training samples: {X_train.shape[0]}")
print(f"Test samples: {X_test.shape[0]}")
print(f"Features: {X_train.shape[1]}")
```

1.4.3 Step 3: Train Model

```python
# Try multiple algorithms
models = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(max_depth=5),
    'Random Forest': RandomForestClassifier(n_estimators=10)
}

results = {}
for name, model in models.items():
    # Train
    model.fit(X_train, y_train)

    # Predict
    y_pred = model.predict(X_test)

    # Evaluate
    accuracy = accuracy_score(y_test, y_pred)
    results[name] = accuracy
    print(f"{name}: {accuracy:.2%}")
```

1.4.4 Step 4: Evaluate Best Model

```
# Get best model
best_model_name = max(results, key=results.get)
best_model = models[best_model_name]

# Detailed evaluation
y_pred = best_model.predict(X_test)
print(classification_report(y_test, y_pred, target_names=['Not Spam', 'Spam'])
    )

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title(f'Confusion Matrix - {best_model_name}')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

1.5 Exercise 2: Customer Segmentation (Unsupervised Clustering)

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

# Generate synthetic customer data
np.random.seed(42)
n_customers = 300

# Features: [recency, frequency, monetary_value]
customers = np.random.randn(n_customers, 3)
customers[:100, :] += [2, 2, 2]  # High-value segment
customers[100:200, :] += [-2, -2, -2]  # Low-value segment

# Standardize
scaler = StandardScaler()
customers_scaled = scaler.fit_transform(customers)

# Find optimal K using elbow method
inertias = []
K_range = range(2, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(customers_scaled)
    inertias.append(kmeans.inertia_)

# Plot elbow
plt.plot(K_range, inertias, 'bo-')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()

# Fit final model
k_optimal = 3
kmeans = KMeans(n_clusters=k_optimal, random_state=42)
clusters = kmeans.fit_predict(customers_scaled)

# Visualize (2D PCA projection)
```

```
pca = PCA(n_components=2)
customers_2d = pca.fit_transform(customers_scaled)

plt.scatter(customers_2d[:, 0], customers_2d[:, 1], c=clusters, cmap='viridis'
    )
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            marker='X', s=200, c='red', edgecolors='black', label='Centroids')
plt.title('Customer Segments')
plt.legend()
plt.show()

# Analyze segments
for i in range(k_optimal):
    segment = customers[clusters == i]
    print(f"\nSegment {i}:")
    print(f"  Size: {len(segment)}")
    print(f"  Avg Recency: {segment[:, 0].mean():.2f}")
    print(f"  Avg Frequency: {segment[:, 1].mean():.2f}")
    print(f"  Avg Monetary: {segment[:, 2].mean():.2f}")
```

1.6 Exercise 3: Complete ML Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score, GridSearchCV

# Build pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier(random_state=42))
])

# Hyperparameter tuning
param_grid = {
    'classifier__n_estimators': [10, 50, 100],
    'classifier__max_depth': [3, 5, 10, None],
    'classifier__min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print(f"Best parameters: {grid_search.best_params_}")
print(f"Best CV score: {grid_search.best_score_:.2%}")

# Test set performance
best_pipeline = grid_search.best_estimator_
test_score = best_pipeline.score(X_test, y_test)
print(f"Test score: {test_score:.2%}")
```

1.7 Key Implementation Patterns

1.7.1 1. Train/Test Split (ALWAYS)

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y  # Preserve class balance
)
```

1.7.2 2. Feature Scaling (for distance-based algorithms)

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)  # Use training stats!
```

1.7.3 3. Cross-Validation (better than single split)

```
scores = cross_val_score(model, X, y, cv=5)
print(f"CV Accuracy: {scores.mean():.2%} (+/- {scores.std():.2%})")
```

1.7.4 4. Model Persistence

```
import joblib

# Save model
joblib.dump(best_model, 'spam_classifier.pkl')

# Load model
loaded_model = joblib.load('spam_classifier.pkl')
```

1.8 Common Issues and Solutions

1.8.1 Issue 1: Imbalanced Classes

```
from sklearn.utils.class_weight import compute_class_weight

# Auto-balance weights
class_weights = compute_class_weight('balanced', classes=np.unique(y_train), y
    =y_train)
model = LogisticRegression(class_weight='balanced')
```

1.8.2 Issue 2: High Dimensionality

```
from sklearn.decomposition import PCA

# Reduce dimensions
pca = PCA(n_components=0.95)  # Keep 95% variance
X_reduced = pca.fit_transform(X)
```

1.8.3 Issue 3: Missing Data

```python
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
```

1.9 Practice Exercises

1. **Iris Classification**: Load iris dataset, train 3 models, compare performance
2. **Boston Housing**: Predict prices using regression (continuous target)
3. **Digits Recognition**: CNN on MNIST dataset
4. **Text Classification**: 20 newsgroups multi-class classification
5. **Anomaly Detection**: Isolation Forest on credit card fraud dataset

---

1.10 Next Steps

- Kaggle competitions (Titanic, House Prices)
- Scikit-learn documentation (excellent examples)
- Week 00b: Deep dive into supervised algorithms