Machine Learning for Bioinformatics

```
Protein Function Prediction Script:

This program is designed to help a computer learn how to predict what a protein does, based on some of its characteristics.

First, it creates a fake dataset of 1,000 proteins with different features like how long each protein is, how much it repels water (calle

Each protein is also assigned a function such as being an enzyme, a transporter, part of a cell’s structure, or involved in signaling.

The goal is for the computer to learn the connection between these features and the protein’s function.

Next, the program prepares this data so that it can be used for machine learning.

It converts the structure types into numbers (since computers do not understand text easily), and splits the data into two groups—one for

It then uses three different types of learning techniques (Random Forest, SVM, and Logistic Regression) to train the computer and see how
After testing the models, the program compares their performance and saves detailed results about how well each method worked.

Finally, it checks which features (like sequence length or charge) were the most useful in making accurate predictions.

This helps us understand what traits ( Quality/Feature) are most important for determining a protein’s role.

All of the results, including charts and reports, are saved to the computer.
```

```python
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import model_selection, metrics, svm
from sklearn.ensemble import RandomForestClassifier

# Create output directory
output_dir = "D:\\LTU\\PU\\output"
os.makedirs(output_dir, exist_ok=True)

# ======================================================================
# 1. Generate Synthetic Protein Data
# ======================================================================
print("Generating synthetic protein data...")

np.random.seed(42)  # For reproducibility

# Create realistic protein features:
# - sequence_length: Length of protein sequence
# - hydrophobicity: Measure of water-repelling tendency
# - charge: Electrical charge of protein
# - secondary_structure: Protein folding pattern
# - function: What the protein does
protein_data = {
    'sequence_length': np.random.randint(50, 500, 1000),
    'hydrophobicity': np.random.normal(0.5, 0.15, 1000).clip(0.1, 0.9),
    'charge': np.random.normal(0, 0.3, 1000).clip(-1, 1),
    'secondary_structure': np.random.choice(['alpha', 'beta', 'mixed'], 1000, p=[0.4, 0.3, 0.3]),
    'function': np.random.choice(['enzyme', 'transporter', 'structural', 'signaling'],
                               1000, p=[0.4, 0.3, 0.2, 0.1])
}

protein_df = pd.DataFrame(protein_data)

# Add realistic correlations:
# Alpha helices tend to be more hydrophobic
protein_df.loc[protein_df['secondary_structure'] == 'alpha', 'hydrophobicity'] *= 1.1
# Enzymes often have slight negative charge
protein_df.loc[protein_df['function'] == 'enzyme', 'charge'] = np.random.normal(-0.2, 0.2, sum(protein_df['function'] == 'enzyme')).clip(

# Save data
protein_df.to_csv(os.path.join(output_dir, "protein_data.csv"), index=False)
print(f"- Saved protein data to {output_dir}/protein_data.csv")

# ======================================================================
# 2. Prepare Data for Machine Learning
# ======================================================================
print("\nPreparing data for machine learning...")

# Select features and target
X = protein_df[['sequence_length', 'hydrophobicity', 'charge']]
y = protein_df['function']

# Convert secondary structure to numerical values (one-hot encoding)
```

```
secondary_struct_encoded = pd.get_dummies(protein_df['secondary_structure'], prefix='struct')
X = pd.concat([X, secondary_struct_encoded], axis=1)

# Split data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# =====================================================================
# 3. Train and Evaluate Models
# =====================================================================
print("\nTraining and evaluating models...")

# Define classifiers to compare
classifiers = {
    "Random Forest": RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42),
    "SVM": svm.SVC(kernel='rbf', probability=True, random_state=42),
    "Logistic Regression": svm.LinearSVC(max_iter=10000, random_state=42)
}

results = []
for name, clf in classifiers.items():
    print(f"\nTraining {name}...")

    # Train model
    clf.fit(X_train, y_train)

    # Make predictions
    y_pred = clf.predict(X_test)

    # Calculate metrics
    accuracy = metrics.accuracy_score(y_test, y_pred)
    f1 = metrics.f1_score(y_test, y_pred, average='weighted')

    # Store results
    results.append({'Model': name, 'Accuracy': accuracy, 'F1 Score': f1})

    # Print performance
    print(f"{name} Performance:")
    print(f"- Accuracy: {accuracy:.3f}")
    print(f"- F1 Score: {f1:.3f}")

    # Save detailed classification report
    report = metrics.classification_report(y_test, y_pred, output_dict=True)
    pd.DataFrame(report).transpose().to_csv(os.path.join(output_dir, f"{name.lower().replace(' ', '_')}_report.csv"))

# Save comparison results
results_df = pd.DataFrame(results)
results_df.to_csv(os.path.join(output_dir, "model_comparison.csv"), index=False)

# =====================================================================
# 4. Analyze Feature Importance (using Random Forest)
# =====================================================================
print("\nAnalyzing feature importance...")

# Train Random Forest model
rf_model = RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42)
rf_model.fit(X_train, y_train)

# Plot feature importance
plt.figure(figsize=(10, 6))
importances = rf_model.feature_importances_
sorted_idx = importances.argsort()[::-1]  # Sort in descending order

plt.barh(range(X_train.shape[1]), importances[sorted_idx], align='center')
plt.yticks(range(X_train.shape[1]), X_train.columns[sorted_idx])
plt.xlabel("Feature Importance")
plt.title("Random Forest Feature Importance")
plt.tight_layout()
plt.savefig(os.path.join(output_dir, "feature_importance.png"), dpi=300)
plt.close()

print("\nANALYSIS COMPLETE! All outputs saved to:", output_dir)
```

```
Generating synthetic protein data...
- Saved protein data to D:\LTU\PU\output/protein_data.csv

Preparing data for machine learning...

Training and evaluating models...

Training Random Forest...
Random Forest Performance:
- Accuracy: 0.465
- F1 Score: 0.423

Training SVM...
SVM Performance:
- Accuracy: 0.410
- F1 Score: 0.238
C:\Users\LENOVO\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\LENOVO\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\LENOVO\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\LENOVO\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\LENOVO\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\LENOVO\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

Training Logistic Regression...
Logistic Regression Performance:
- Accuracy: 0.515
- F1 Score: 0.424

Analyzing feature importance...

ANALYSIS COMPLETE! All outputs saved to: D:\LTU\PU\output
```

Cancer Subtype Classification Script:

This program helps a computer learn how to identify different types of cancer based on data from genes.

Imagine doctors have information from hundreds of patients, showing how active certain genes are.

This data is used to figure out which type of cancer each patient has, such as Luminal A, Basal, or HER2-enriched.

The program begins by creating a fake (but realistic) set of patient data that includes measurements for 100 genes across 500 patients.

It also assigns a cancer subtype to each patient. This data is saved for future analysis.

Next, the program prepares the data so the computer can understand it.

It splits the data into a training set (for learning) and a test set (to check how well it learned).

It also scales the numbers so they are easier for the computer to work with.

Then, it teaches the computer using three different learning techniques: Random Forest, SVM (Support Vector Machine), and Logistic Regre

Each one tries to learn the connection between gene activity and cancer subtype.

After learning, the computer is tested to see how accurately it can guess a cancer subtype it hasn’t seen before.

The program prints and saves results that show how well each method worked and includes charts that make it easier to understand.

Finally, the program reduces the complex gene data into just two main components using a technique called PCA (Principal Component Analy

It plots the patients on a 2D graph where you can see how clearly the cancer subtypes are separated.

This helps us visually understand how gene activity patterns relate to cancer types.

All results, charts, and reports are saved to a folder.

```python
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets, model_selection, preprocessing, metrics, svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA

# Create output directory
output_dir = "D:\LTU\PU\output"
os.makedirs(output_dir, exist_ok=True)

# ======================================================================
# 1. Generate Synthetic Gene Expression Data
# ======================================================================
print("Generating synthetic gene expression data...")

np.random.seed(42)  # For reproducibility
```

```
# Create realistic gene expression data with:
# - 500 samples (patients)
# - 100 genes measured
# - 3 cancer subtypes
X, y = datasets.make_classification(
    n_samples=500,
    n_features=100,
    n_classes=3,
    n_informative=15,  # Only 15 genes actually matter for classification
    n_clusters_per_class=2,  # More complex structure
    class_sep=1.5,  # Clear separation between classes
    random_state=42
)

# Create meaningful names
gene_names = [f"GENE_{i:03d}" for i in range(X.shape[1])]
subtype_names = ["Luminal_A", "Basal", "HER2_enriched"]
subtype_colors = {"Luminal_A": "blue", "Basal": "red", "HER2_enriched": "purple"}

# Create and save dataset
cancer_df = pd.DataFrame(X, columns=gene_names)
cancer_df['subtype'] = pd.Series(y).map(dict(enumerate(subtype_names)))
cancer_df.to_csv(os.path.join(output_dir, "expression_data.csv"), index=False)
print(f"- Saved gene expression data to {output_dir}/expression_data.csv")

# ======================================================================
# 2. Prepare Data for Machine Learning
# ======================================================================
print("\nPreparing data for machine learning...")

# Separate features (X) and target (y)
X = cancer_df.drop('subtype', axis=1).values
y = cancer_df['subtype'].values

# Split data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# Standardize features (mean=0, variance=1)
scaler = preprocessing.StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# ======================================================================
# 3. Train and Evaluate Models
# ======================================================================
print("\nTraining and evaluating models...")

# Define classifiers to compare
classifiers = {
    "Random Forest": RandomForestClassifier(n_estimators=200, max_depth=10, random_state=42),
    "SVM": svm.SVC(kernel='rbf', probability=True, random_state=42),
    "Logistic Regression": svm.LinearSVC(multi_class='ovr', max_iter=10000, random_state=42)
}

results = []
for name, clf in classifiers.items():
    print(f"\nTraining {name}...")

    # Train model
    clf.fit(X_train, y_train)

    # Make predictions
    y_pred = clf.predict(X_test)

    # Calculate metrics
    accuracy = metrics.accuracy_score(y_test, y_pred)
    f1 = metrics.f1_score(y_test, y_pred, average='weighted')

    # Store results
    results.append({'Model': name, 'Accuracy': accuracy, 'F1 Score': f1})

    # Print performance
    print(f"{name} Performance:")
    print(f"- Accuracy: {accuracy:.3f}")
    print(f"- F1 Score: {f1:.3f}")

    # Save classification report
    report = metrics.classification_report(y_test, y_pred, target_names=subtype_names, output_dict=True)
    pd.DataFrame(report).transpose().to_csv(os.path.join(output_dir, f"{name.lower().replace(' ', '_')}_report.csv"))
    # Create and save confusion matrix
    cm = metrics.confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8,6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=subtype_names, yticklabels=subtype_names)
    plt.title(f"{name} Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.savefig(os.path.join(output_dir, f"{name.lower().replace(' ', '_')}_confusion_matrix.png"), dpi=300)
    plt.close()

# Save comparison results
results_df = pd.DataFrame(results)
results_df.to_csv(os.path.join(output_dir, "model_comparison.csv"), index=False)

# ======================================================================
# 4. Visualize Data with PCA
# ======================================================================
print("\nCreating PCA visualization...")

# Reduce dimensions to 2D for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)  # Use original (unscaled) X for visualization

plt.figure(figsize=(10, 8))
for subtype, color in subtype_colors.items():
    mask = (cancer_df['subtype'] == subtype)
    plt.scatter(X_pca[mask, 0], X_pca[mask, 1], color=color, label=subtype, alpha=0.6)
plt.title("PCA of Gene Expression Data")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend()
plt.savefig(os.path.join(output_dir, "pca_visualization.png"), dpi=300)
plt.close()

print("\nANALYSIS COMPLETE! All outputs saved to:", output_dir)
```

```
<>:11: SyntaxWarning: invalid escape sequence '\L'
<>:11: SyntaxWarning: invalid escape sequence '\L'
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_7764\1060015529.py:11: SyntaxWarning: invalid escape sequence '\L'
  output_dir = "D:\LTU\PU\output"
Generating synthetic gene expression data...
- Saved gene expression data to D:\LTU\PU\output/expression_data.csv

Preparing data for machine learning...

Training and evaluating models...

Training Random Forest...
Random Forest Performance:
- Accuracy: 0.750
- F1 Score: 0.747

Training SVM...
SVM Performance:
- Accuracy: 0.730
- F1 Score: 0.728

Training Logistic Regression...
Logistic Regression Performance:
- Accuracy: 0.690
- F1 Score: 0.681

Creating PCA visualization...

ANALYSIS COMPLETE! All outputs saved to: D:\LTU\PU\output
```

Start coding or generate with AI.