ANN scripts

(Note: Only a certain portion of code is included.)

Model Training

```
"""Train and evaluate multiple models"""
models = {
'Neural Network': {
'model': MLPClassifier(random_state=42),
'params': {
'hidden_layer_sizes': [(32,), (64,), (32, 32), (64, 32), (64, 64)],  # Layer configurations
'activation': ['relu', 'tanh'],  # Activation functions
'alpha': [0.0001, 0.001, 0.01, 0.1],  # L2 regularization term
'learning_rate': ['constant', 'adaptive'],  # Learning rate schedule
'solver': ['adam', 'sgd'],  # Optimization algorithms
'batch_size': [16, 32, 64],  # Batch sizes
'max_iter': [500, 1000],  # Maximum iterations,
    'early_stopping': [True],  # Added early stopping
        'validation_fraction': [0.2]

}
}
}
```

```
def _plot_confusion_matrix(y_true, y_pred, model_name):
    """Plot confusion matrix using plotly"""
    cm = confusion_matrix(y_true, y_pred)

    fig = go.Figure(data=go.Heatmap(
        z=cm,
        x=['Predicted 0', 'Predicted 1'],
        y=['Actual 0', 'Actual 1'],
        text=cm,
        texttemplate='%{text}',
        textfont={'size': 20},
        colorscale='Blues'
    ))

    fig.update_layout(
        title=f'Confusion Matrix - {model_name}',
        xaxis_title='Predicted Label',
        yaxis_title='True Label'
    )
    fig.show()
```

```python
def _get_feature_importance( model, model_name, feature_names):
    """Extract feature importance from model if available"""
    if hasattr(model, 'feature_importances_'):
        return pd.Series(model.feature_importances_, index=feature_names)
    elif model_name == 'Logistic Regression':
        return pd.Series(abs(model.coef_[0]), index=feature_names)
    elif model_name == 'SVM' and model.kernel == 'linear':
        return pd.Series(abs(model.coef_[0]), index=feature_names)
    else:
        return None

# Train and evaluate each model
for name, model_info in models.items():
    print(f"\nTraining {name}...")

    # Use RandomizedSearchCV for efficiency
    random_search = RandomizedSearchCV(
        model_info['model'],
        model_info['params'],
        n_iter=20,
        cv=5,
        scoring=['accuracy', 'roc_auc', 'f1'],
        refit='roc_auc',
        n_jobs=-1,
        random_state=42
    )

    # Fit model
    random_search.fit(X_train, y_train)

    # Store results
    results[name] = {
        'model': random_search.best_estimator_,
        'best_params': random_search.best_params_,
        'cv_results': random_search.cv_results_,
        'train_score': random_search.score(X_train, y_train),
        'test_score': random_search.score(X_test, y_test),
        'predictions': random_search.predict(X_test),
        'probabilities': random_search.predict_proba(X_test)[:, 1],
        'feature_importance': _get_feature_importance(
            random_search.best_estimator_, name, X_train.columns)
    }

    # Print results
    print(f"\nBest parameters: {random_search.best_params_}")
    print("\nClassification Report:")
    print(classification_report(y_test, results[name]['predictions']))
```

```
    # Plot confusion matrix
    _plot_confusion_matrix(y_test, results[name]['predictions'], name)

# # Train and evaluate each model
# for name, model_info in models.items():
#     print(f"\nTraining {name}...")

#     # Use RandomizedSearchCV for efficiency
#     random_search = RandomizedSearchCV(
#         model_info['model'],
#         model_info['params'],
#         n_iter=20,
#         cv=5,
#         scoring=['accuracy', 'roc_auc', 'f1'],
#         refit='roc_auc',
#         n_jobs=-1,
#         random_state=42
#     )

#     # Fit model
#     random_search.fit(X_train, y_train)

#     # Store results (don't use test data here during training)
#     results[name] = {
#         'model': random_search.best_estimator_,
#         'best_params': random_search.best_params_,
#         'cv_results': random_search.cv_results_,
#         'train_score': random_search.score(X_train, y_train),
#         'predictions': random_search.predict(X_test),  # Only predict on test data after
training
#         'probabilities': random_search.predict_proba(X_test)[:, 1],
#         'feature_importance': _get_feature_importance(
#             random_search.best_estimator_, name, X_train.columns)
#     }

#     # Print results
#     print(f"\nBest parameters: {random_search.best_params_}")
#     print("\nClassification Report:")
#     print(classification_report(y_test, results[name]['predictions']))

#     # Plot confusion matrix
#     _plot_confusion_matrix(y_test, results[name]['predictions'], name)

# def train_and_evaluate_models(models, X_train, X_test, y_train, y_test,
random_state=42):
#     """
```

 # Train and evaluate multiple models with proper cross-validation and performance metrics.