Diabetes Outcome Predictions from Related Health Measurements

Class Introduction to Data Science 1/2022

Lecturer: Lect. Dr. Petchara Pattarabanjird

Phychat Udomsopagit 6339046

```
[1]: import numpy as np
     import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
```

Data Preprocessing

Import dataset downloaded from Kaggle

- Kaggle's archive of Kaggle's and Organizer's Datasets
- Pandas, 11.31 years old
- Pandas Documentation

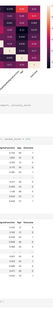
```
[2]: !kaggle datasets download -d mathchi/diabetes-data-set
     !ls -lh diabetes*
     !unzip diabetes.csv
     !head -5 diabetes.csv
```

```
Dataset URL: https://www.kaggle.com/datasets/mathchi/diabetes-data-set
-rw-r--r-- 1 root root 23K Jan 27 07:41 diabetes.csv.zip
Archive:  diabetes.csv.zip
  inflating: diabetes.csv
Pregnancies,Glucose,BloodPressure,SkinThickness,Insulin,BMI,DiabetesPedigreeFunction,Age,Outcome
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1
```

```
[3]: df = pd.read_csv('diabetes.csv')
     df.shape
     df.head()
     df.info()
     df.describe()
     df.isnull().sum
     df.corr()
     import matplotlib as mpl
     import matplotlib.pyplot as plt
     import seaborn as sns
     plt.rcParams['figure.dpi'] = 150
     plt.rcParams['font.family'] = 'Times New Roman'
     plt.rcParams['font.size'] = 12
     plt.rcParams['axes.unicode_minus'] = False
     sns.heatmap(df.corr(), annot=True, cmap='Blues')
     plt.xticks(rotation=45)
     plt.yticks(rotation=0)
     plt.savefig('heatmap.jpg')
     plt.show()
```

Visualize feature correlation of the dataset

```
[4]: from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.tree import DecisionTreeClassifier
     X = df.drop('Outcome', axis=1)
     y = df['Outcome']
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
     scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)
     print("X_train.shape:", X_train.shape)
     print("X_test.shape:", X_test.shape)
     print("y_train.shape:", y_train.shape)
     print("y_test.shape:", y_test.shape)
```




```
[5]: X_train.shape, X_test.shape, y_train.shape, y_test.shape
     X_train[:5]
     X_test[:5]
     y_train[:5]
     y_test[:5]
```

```
X_train.shape: (614, 8)
X_test.shape: (154, 8)
y_train.shape: (614,)
y_test.shape: (154,)
```

Split dataset into train and test sets

- Goal0s to be randomly balance outcome of the patients
- Test set size = 20%

```
[6]: clf = RandomForestClassifier(n_estimators=100, random_state=42)
     clf.fit(X_train, y_train)
     y_pred = clf.predict(X_test)
     from sklearn.metrics import classification_report, confusion_matrix
     print(classification_report(y_test, y_pred))
     print(confusion_matrix(y_test, y_pred))
```

Assign input features and output to X and y

```
[7]: from sklearn.ensemble import RandomForestClassifier
     clf = RandomForestClassifier(n_estimators=100, random_state=42)
     clf.fit(X_train, y_train)
     feature_importances = clf.feature_importances_
     print(feature_importances)
```

Normalize the input value of feature

- Using StandardScaler

```
[8]: from sklearn.tree import DecisionTreeClassifier
     clf = DecisionTreeClassifier(random_state=42)
     clf.fit(X_train, y_train)
     feature_importances = clf.feature_importances_
     print(feature_importances)
```

Building Model, Implementation and Evaluation

Random forest and decision tree

Define function to train random forest and decision tree model

- Random forest
  - n_estimator = 100
  - max depth=10
- Decision tree, importance function to determine features score

```
[9]: from sklearn.ensemble import RandomForestClassifier
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.metrics import classification_report, confusion_matrix
     def train_model(X_train, y_train):
         rf = RandomForestClassifier(n_estimators=100, random_state=42)
         rf.fit(X_train, y_train)
         dt = DecisionTreeClassifier(random_state=42)
         dt.fit(X_train, y_train)
         return rf, dt
     def print_metrics(model, X_test, y_test):
         y_pred = model.predict(X_test)
         print(classification_report(y_test, y_pred))
         print(confusion_matrix(y_test, y_pred))
     rf, dt = train_model(X_train, y_train)
     print("Random Forest:")
     print_metrics(rf, X_test, y_test)
     print("Decision Tree:")
     print_metrics(dt, X_test, y_test)
```

Train the model

```
[10]: from sklearn.model_selection import cross_val_score, GridSearchCV
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.tree import DecisionTreeClassifier
      rf = RandomForestClassifier(n_estimators=100, random_state=42)
      dt = DecisionTreeClassifier(random_state=42)
      rf_cv_scores = cross_val_score(rf, X_train, y_train, cv=5)
      dt_cv_scores = cross_val_score(dt, X_train, y_train, cv=5)
      print("Random Forest cross-validation scores:", rf_cv_scores)
      print("Decision Tree cross-validation scores:", dt_cv_scores)
      rf_param_grid = {'n_estimators': [100, 200, 300], 'max_depth': [None, 10, 20, 30]}
      dt_param_grid = {'max_depth': [None, 10, 20, 30]}
      rf_grid_search = GridSearchCV(rf, rf_param_grid, cv=5)
      rf_grid_search.fit(X_train, y_train)
      print("Random Forest best parameters:", rf_grid_search.best_params_)
      dt_grid_search = GridSearchCV(dt, dt_param_grid, cv=5)
      dt_grid_search.fit(X_train, y_train)
      print("Decision Tree best parameters:", dt_grid_search.best_params_)
```

```
Random Forest cross-validation scores: [0.71544715 0.74796748 0.76422764 0.76422764 0.77868852]
Decision Tree cross-validation scores: [0.64227642 0.69105691 0.69105691 0.69105691 0.72131147]
Random Forest best parameters: {'n_estimators': 100, 'max_depth': None}
Decision Tree best parameters: {'max_depth': None}
```

Effect of number of feature on accuracy score of the models

```
[11]: from sklearn.ensemble import RandomForestClassifier
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.metrics import classification_report, confusion_matrix
      rf = RandomForestClassifier(n_estimators=100, random_state=42)
      dt = DecisionTreeClassifier(random_state=42)
      rf.fit(X_train, y_train)
      dt.fit(X_train, y_train)
      rf_y_pred = rf.predict(X_test)
      dt_y_pred = dt.predict(X_test)
      print("Random Forest:")
      print(classification_report(y_test, rf_y_pred))
      print(confusion_matrix(y_test, rf_y_pred))
      print("Decision Tree:")
      print(classification_report(y_test, dt_y_pred))
      print(confusion_matrix(y_test, dt_y_pred))
```

```
Random Forest:
              precision    recall  f1-score   support

           0       0.81      0.90      0.85       100
           1       0.73      0.57      0.64        54

    accuracy                           0.79       154
   macro avg       0.77      0.73      0.74       154
weighted avg       0.78      0.79      0.78       154

[[90 10]
 [23 31]]
Decision Tree:
              precision    recall  f1-score   support

           0       0.79      0.79      0.79       100
           1       0.63      0.63      0.63        54

    accuracy                           0.73       154
   macro avg       0.71      0.71      0.71       154
weighted avg       0.73      0.73      0.73       154

[[79 21]
 [20 34]]
```

```
[12]: import numpy as np
      import matplotlib as mpl
      import matplotlib.pyplot as plt
      from sklearn.ensemble import RandomForestClassifier
      rf = RandomForestClassifier(n_estimators=100, random_state=42)
      rf.fit(X_train, y_train)
      feature_importances = rf.feature_importances_
      sorted_indices = np.argsort(feature_importances)[::-1]
      plt.rcParams['font.family'] = 'Times New Roman'
      plt.rcParams['font.size'] = 12
      plt.rcParams['axes.unicode_minus'] = False
      plt.rcParams['figure.dpi'] = 150
      plt.bar(range(X_train.shape[1]), feature_importances[sorted_indices], color='#1f77b4')
      plt.xticks(range(X_train.shape[1]), X.columns[sorted_indices], rotation=45)
      plt.ylabel('Feature Importance')
      plt.title('Feature Importance')
      plt.savefig('feature_importance.jpg')
      plt.show
```

Visualize the feature score against the model accuracy

```
[13]: from sklearn.ensemble import RandomForestClassifier
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.metrics import classification_report, confusion_matrix
      rf = RandomForestClassifier(n_estimators=100, random_state=42)
      dt = DecisionTreeClassifier(random_state=42)
      rf.fit(X_train, y_train)
      dt.fit(X_train, y_train)
      rf_y_pred = rf.predict(X_test)
      dt_y_pred = dt.predict(X_test)
      print("Random Forest:")
      print(classification_report(y_test, rf_y_pred))
      print(confusion_matrix(y_test, rf_y_pred))
      print("Decision Tree:")
      print(classification_report(y_test, dt_y_pred))
      print(confusion_matrix(y_test, dt_y_pred))
      import numpy as np
      import matplotlib as mpl
      import matplotlib.pyplot as plt
      from sklearn.ensemble import RandomForestClassifier
      rf = RandomForestClassifier(n_estimators=100, random_state=42)
      rf.fit(X_train, y_train)
      feature_importances = rf.feature_importances_
      sorted_indices = np.argsort(feature_importances)[::-1]
      plt.rcParams['font.family'] = 'Times New Roman'
      plt.rcParams['font.size'] = 12
      plt.rcParams['axes.unicode_minus'] = False
      plt.rcParams['figure.dpi'] = 150
      plt.bar(range(X_train.shape[1]), feature_importances[sorted_indices], color='#1f77b4')
      plt.xticks(range(X_train.shape[1]), X.columns[sorted_indices], rotation=45)
      plt.ylabel('Feature Importance')
      plt.title('Feature Importance')
      plt.savefig('feature_importance.jpg')
      plt.show
```

```
Random Forest:
              precision    recall  f1-score   support

           0       0.81      0.90      0.85       100
           1       0.73      0.57      0.64        54

    accuracy                           0.79       154
   macro avg       0.77      0.73      0.74       154
weighted avg       0.78      0.79      0.78       154

[[90 10]
 [23 31]]
Decision Tree:
              precision    recall  f1-score   support

           0       0.79      0.79      0.79       100
           1       0.63      0.63      0.63        54

    accuracy                           0.73       154
   macro avg       0.71      0.71      0.71       154
weighted avg       0.73      0.73      0.73       154

[[79 21]
 [20 34]]
```

Implement model to test dataset to evaluate the model

```
[14]: from sklearn.ensemble import RandomForestClassifier
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.metrics import classification_report, confusion_matrix
      rf = RandomForestClassifier(n_estimators=100, random_state=42)
      dt = DecisionTreeClassifier(random_state=42)
      rf.fit(X_train, y_train)
      dt.fit(X_train, y_train)
      rf_y_pred = rf.predict(X_test)
      dt_y_pred = dt.predict(X_test)
      print("Random Forest:")
      print(classification_report(y_test, rf_y_pred))
      print(confusion_matrix(y_test, rf_y_pred))
      print("Decision Tree:")
      print(classification_report(y_test, dt_y_pred))
      print(confusion_matrix(y_test, dt_y_pred))
```

```
Random Forest:
              precision    recall  f1-score   support

           0       0.81      0.90      0.85       100
           1       0.73      0.57      0.64        54

    accuracy                           0.79       154
   macro avg       0.77      0.73      0.74       154
weighted avg       0.78      0.79      0.78       154

[[90 10]
 [23 31]]
Decision Tree:
              precision    recall  f1-score   support

           0       0.79      0.79      0.79       100
           1       0.63      0.63      0.63        54

    accuracy                           0.73       154
   macro avg       0.71      0.71      0.71       154
weighted avg       0.73      0.73      0.73       154

[[79 21]
 [20 34]]
```

Function to visualize confusion matrix

```
[15]: def plot_confusion_matrix(y_true, y_pred, classes=None):
          from sklearn.metrics import confusion_matrix
          cm = confusion_matrix(y_true, y_pred)
          if classes is None:
              classes = ['No diabetes', 'Diabetes']
          fig, ax = plt.subplots(figure=fig, dpi=150)
          im = ax.imshow(cm, interpolation='nearest', cmap='Blues')
          ax.set_xticks(np.arange(len(classes)))
          ax.set_yticks(np.arange(len(classes)))
          ax.set_xticklabels(classes, rotation=45, ha='right')
          ax.set_yticklabels(classes)
          ax.set_ylabel('True')
          ax.set_xlabel('Predicted')
          for i in range(len(classes)):
              for j in range(len(classes)):
                  text = ax.text(j, i, cm[i, j], ha='center', va='center', color='white' if cm[i, j] > cm.max()/2.
          fig.colorbar(im, ax=ax)
          plt.tight_layout()
          return fig, ax
      plot_confusion_matrix(y_test, rf_y_pred, classes=['No diabetes', 'Diabetes'])
      plot_confusion_matrix(y_test, dt_y_pred, classes=['No diabetes', 'Diabetes'])
```

Calculate score, confusion matrix and visualize confusion matrix

```
[16]: from sklearn.ensemble import RandomForestClassifier, DecisionTreeClassifier
      from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall
      print("Random Forest:")
      print(classification_report(y_test, rf_y_pred))
      print(confusion_matrix(y_test, rf_y_pred))
      print(accuracy_score(y_test, rf_y_pred))
      print(precision_score(y_test, rf_y_pred))
      print(recall_score(y_test, rf_y_pred))
      print("Decision Tree:")
      print(classification_report(y_test, dt_y_pred))
      print(confusion_matrix(y_test, dt_y_pred))
      print(accuracy_score(y_test, dt_y_pred))
      print(precision_score(y_test, dt_y_pred))
      print(recall_score(y_test, dt_y_pred))
      import numpy as np
      import matplotlib as mpl
      import matplotlib.pyplot as plt
      from sklearn.ensemble import RandomForestClassifier
      rf = RandomForestClassifier(n_estimators=100, random_state=42)
      rf.fit(X_train, y_train)
      feature_importances = rf.feature_importances_
      sorted_indices = np.argsort(feature_importances)[::-1]
      plt.rcParams['font.family'] = 'Times New Roman'
      plt.rcParams['font.size'] = 12
      plt.rcParams['axes.unicode_minus'] = False
      plt.rcParams['figure.dpi'] = 150
      plt.bar(range(X_train.shape[1]), feature_importances[sorted_indices], color='#1f77b4')
      plt.xticks(range(X_train.shape[1]), X.columns[sorted_indices], rotation=45)
      plt.ylabel('Feature Importance')
      plt.title('Feature Importance')
      plt.savefig('feature_importance.jpg')
      plt.show
```

```
Random Forest:
              precision    recall  f1-score   support

           0       0.81      0.90      0.85       100
           1       0.73      0.57      0.64        54

    accuracy                           0.79       154
   macro avg       0.77      0.73      0.74       154
weighted avg       0.78      0.79      0.78       154

[[90 10]
 [23 31]]
0.7857142857142857
0.7
0.5740740740740741
Decision Tree:
              precision    recall  f1-score   support

           0       0.79      0.79      0.79       100
           1       0.63      0.63      0.63        54

    accuracy                           0.73       154
   macro avg       0.71      0.71      0.71       154
weighted avg       0.73      0.73      0.73       154

[[79 21]
 [20 34]]
0.7337662337662337
0.6666666666666666
0.6666666666666666
```

Multi Layer Perceptron (MLP), Neural Network

Import necessary library

```
[17]: import numpy as np
      import matplotlib as mpl
      from sklearn.neural_network import MLPClassifier
      from sklearn.metrics import classification_report, confusion_matrix
      mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500, random_state=42)
      mlp.fit(X_train, y_train)
      mlp_y_pred = mlp.predict(X_test)
      print("MLP:")
      print(classification_report(y_test, mlp_y_pred))
      print(confusion_matrix(y_test, mlp_y_pred))
      import numpy as np
      import matplotlib as mpl
      from sklearn.ensemble import RandomForestClassifier
      rf = RandomForestClassifier(n_estimators=100, random_state=42)
      rf.fit(X_train, y_train)
      feature_importances = rf.feature_importances_
      sorted_indices = np.argsort(feature_importances)[::-1]
      plt.rcParams['font.family'] = 'Times New Roman'
      plt.rcParams['font.size'] = 12
      plt.rcParams['axes.unicode_minus'] = False
      plt.rcParams['figure.dpi'] = 150
      plt.bar(range(X_train.shape[1]), feature_importances[sorted_indices], color='#1f77b4')
      plt.xticks(range(X_train.shape[1]), X.columns[sorted_indices], rotation=45)
      plt.ylabel('Feature Importance')
      plt.title('Feature Importance')
      plt.savefig('feature_importance.jpg')
      plt.show
```

```
MLP:
              precision    recall  f1-score   support

           0       0.76      0.94      0.84       100
           1       0.67      0.31      0.42        54

    accuracy                           0.75       154
   macro avg       0.71      0.62      0.63       154
weighted avg       0.73      0.75      0.71       154

[[94 6]
 [37 17]]
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686:UserWarning:Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't converged.
/usr/local/lib/python3.10/dist-packages/matplotlib/font_manager.py:1332:UserWarning:findfont: Font family ['Times New Roman'] not found. Falling back to DejaVu Sans
```

```
Epoch 19/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 20/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
```

Visualize the model loss value accuracy score on iteration numbers

```
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 2/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 3/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 4/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 5/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 6/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 7/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 8/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 9/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 10/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 11/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 12/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 13/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 14/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 15/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 16/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 17/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 18/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 19/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 20/20
[==============================] - ETA: 0s - loss: 0.5481 - accuracy: 0.6904 - val_loss: 0.5481 - v
Epoch 1/20
```