MINOR PROJECT

Name: D Karthik

Batch: May ML 2

Mentor: Liqzan Manna

Diabetes Case Study

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

- **Step 1:** Let's do a few steps here. Take a look at some of usual summary statistics calculated to accurately match the values to the appropriate key.
- **Step 2**: Since our dataset here is quite clean, we will jump straight into the machine learning. Our goal here is to be able to predict cases of diabetes. First, you need to identify the y vector and X matrix. Then, the following code will divide your dataset into training and test data.
- **Step 3**: In this step, I will show you how to use randomized search, and then you can set up grid searches for the other models in Step 4. However, you will be helping, as I don't remember exactly what each of the hyperparameters in SVMs do. Match each hyperparameter to its corresponding tuning functionality.
- **Step 4**: Now that you have seen how to run a randomized grid search using random forest, try this out for the AdaBoost and SVC classifiers. You might also decide to try out other classifiers that you saw earlier in the lesson to see what works best.
- **Step 6**: Despite the fact that your models here are more difficult to interpret, there are some ways to get an idea of which features are important.

CODE:

Diabetes Prediction.ipynb

File Edit View Insert Runtime Tools Help Last edited on June 22

Comment Share

+ Code + Text

Diabetes Case Study

```
# Import our libraries
import pandas as pd
import numpy as np
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
import matplotlib.pyplot as plt
from sklearn.svm import SVC
import seaborn as sns
sns.set(style="ticks")

import check_file as ch

%matplotlib inline

# Read in our dataset
diabetes = pd.read_csv('diabetes.csv')

# Take a look at the first few rows of the dataset
diabetes.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Step 1: Let's do a few steps here. Take a look at some of usual summary statistics calculated to accurately match the values to the appropriate key in the dictionary below.

Diabetes Prediction.ipynb

File Edit View Insert Runtime Tools Help Last edited on June 22

Comment Share

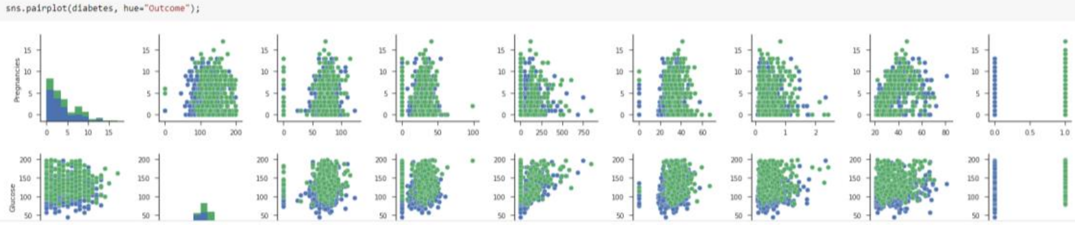
+ Code + Text

Step 1: Let's do a few steps here. Take a look at some of usual summary statistics calculated to accurately match the values to the appropriate key in the dictionary below.


```
# Cells for work
diabetes.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

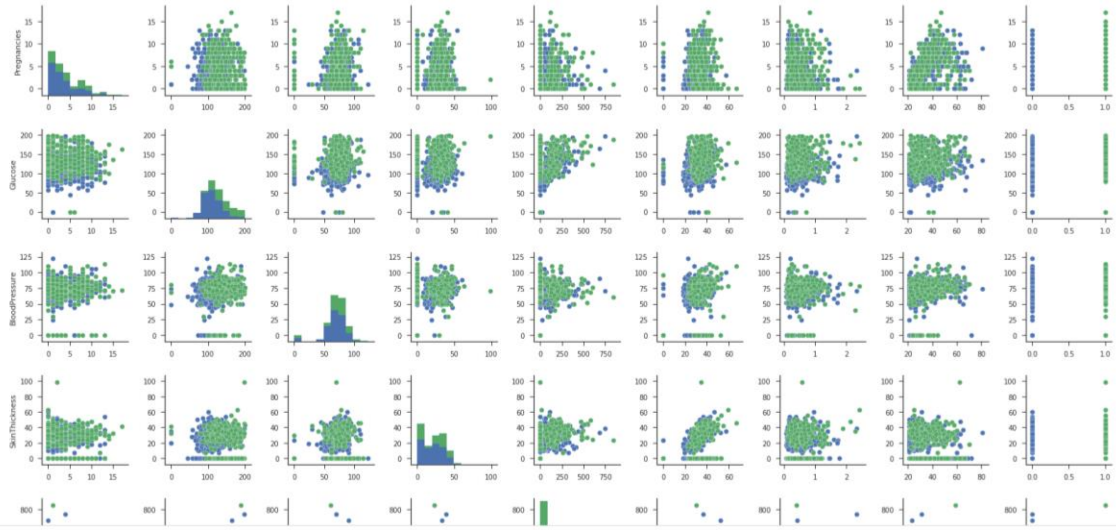
```
sns.pairplot(diabetes, hue="Outcome");
```



Diabetes Prediction.ipynb

File Edit View Insert Runtime Tools Help Last edited on June 22

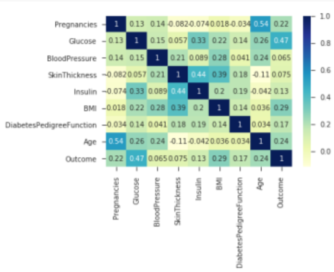
```
sns.pairplot(diabetes, hue="Outcome");
```



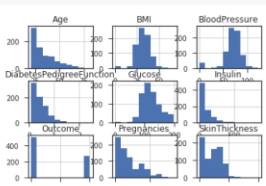
Diabetes Prediction.ipynb

File Edit View Insert Runtime Tools Help Last edited on June 22

```
sns.heatmap(diabetes.corr(), annot=True, cmap="YlGnBu");
```




```
diabetes.hist();
```



Diabetes Prediction.ipynb

File Edit View Insert Runtime Tools Help Last edited on June 22

\+ Code + Text

Comment Share

RAM Disk Editing

0.0 0.5 1.0 0 10 0 50 100

```
[ ] # Possible keys for the dictionary
    a = '0.65'
    b = '0'
    c = 'Age'
    d = '0.35'
    e = 'Glucose'
    f = '0.5'
    g = "More than zero"

    # Fill in the dictionary with the correct values here
    answers_one = {
        'The proportion of diabetes outcomes in the dataset': d,
        'The number of missing data points in the dataset': b,
        'A dataset with a symmetric distribution': e,
        'A dataset with a right-skewed distribution': c,
        'This variable has the strongest correlation with the outcome': e
    }

    # Just to check your answer, don't change this
    ch.check_one(answers_one)
```

Step 2: Since our dataset here is quite clean, we will jump straight into the machine learning. Our goal here is to be able to predict cases of diabetes. First, we need to identify the y vector and X matrix. Then, the following code will divide your dataset into training and test data.

```
[ ] y = diabetes['Outcome']
    X = diabetes[['Pregnancies','Glucose', 'BloodPressure', 'SkinThickness','Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Now we have a training and testing dataset, we need to create some models that and ultimately find the best of them. However, unlike in earlier, where we used the defaults, we can now tune these models to be the very best models they can be.

Step 3: In this step, we use randomized search, and then we can set up grid searches for the other models in Step 4. Match each hyperparameter to its corresponding tuning functionality.

Diabetes Prediction.ipynb

File Edit View Insert Runtime Tools Help Last edited on June 22

\+ Code + Text

Comment Share

RAM Disk Editing

where we used the defaults, we can now tune these models to be the very best models they can be.

Step 3: In this step, we use randomized search, and then we can set up grid searches for the other models in Step 4. Match each hyperparameter to its corresponding tuning functionality.


```
[ ] # build a classifier
    clf_rf = RandomForestClassifier()

    # Set up the hyperparameter search
    param_dist = {"max_depth": [3, None],
                  "n_estimators": list(range(10, 200)),
                  "max_features": list(range(1, X_test.shape[1]+1)),
                  "min_samples_split": list(range(2, 11)),
                  "min_samples_leaf": list(range(1, 11)),
                  "bootstrap": [True, False],
                  "criterion": ["gini", "entropy"]}


    # Run a randomized search over the hyperparameters
    random_search = RandomizedSearchCV(clf_rf, param_distributions=param_dist)

    # Fit the model on the training data
    random_search.fit(X_train, y_train)

    # Make predictions on the test data
    rf_preds = random_search.best_estimator_.predict(X_test)

    ch.print_metrics(y_test, rf_preds, 'random forest')
```

```
Accuracy score for random forest : 0.7402597402597403
Precision score random forest : 0.631578947368421
Recall score random forest : 0.6545454545454545
F1 score random forest : 0.6428571428571428
```

Step 4: Now that we have seen how to run a randomized grid search using random forest,Now we will try this out for the AdaBoost and SVC classifiers. we might also decide to try out other classifiers.what works best.

Diabetes Prediction.ipynb

File Edit View Insert Runtime Tools Help Last edited on June 22

+ Code + Text

Step 4: Now that we have seen how to run a randomized grid search using random forest,Now we will try this out for the AdaBoost and SVC classifiers. we might also decide to try out other classifiers.what works best.

```
[ ] # build a classifier for ada boost
    clf_ada = AdaBoostClassifier()

    # Set up the hyperparameter search
    # look at  setting up your search for n_estimators, learning_rate
    # http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
    param_dist = {"n_estimators": [10, 100, 200, 400],
                  "learning_rate": [0.001, 0.005, .01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1, 2, 10, 20]}


    # Run a randomized search over the hyperparameters
    ada_search = RandomizedSearchCV(clf_ada, param_distributions=param_dist)

    # Fit the model on the training data
    ada_search.fit(X_train, y_train)

    # Make predictions on the test data
    ada_preds = ada_search.best_estimator_.predict(X_test)

    ch.print_metrics(y_test, ada_preds, 'adaboost')
```

```
Accuracy score for adaboost : 0.7597402597402597
Precision score adaboost : 0.6551724137931034
Recall score adaboost : 0.6909090909090909
F1 score adaboost : 0.6725663716814159
```

```
[ ] # build a classifier for support vector machines
    clf_svc = SVC()

    # Set up the hyperparameter search
    # look at setting up your search for C (recommend 0-10 range),
    # kernel, and degree
    # http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
    param_dist = {"C": [0.1, 0.5, 1, 3, 5],
                  "kernel": ['linear', 'rbf']
```

Diabetes Prediction.ipynb

File Edit View Insert Runtime Tools Help Last edited on June 22

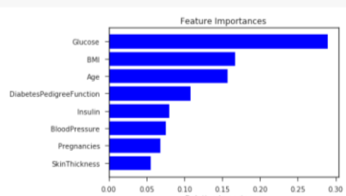
+ Code + Text

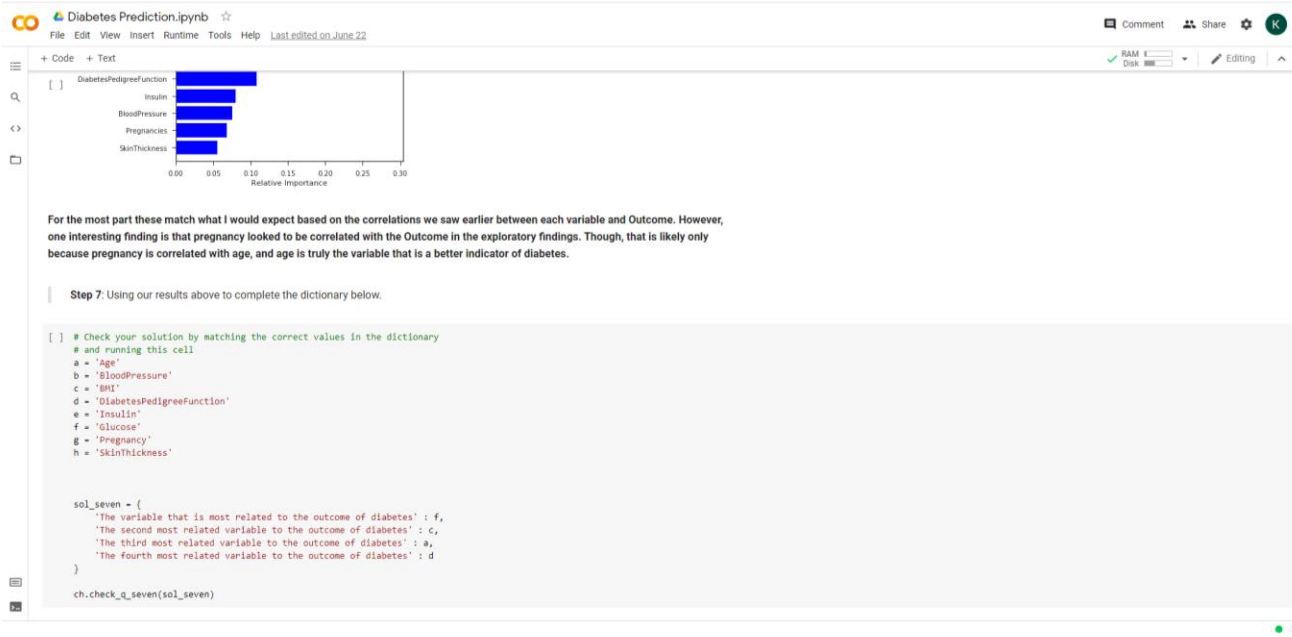
Step 5: Using the test below to see if our best model matched, what we found after running the grid search.

```
[ ] a = 'randomforest'
    b = 'adaboost'
    c = 'supportvector'

    best_model = b # put your best model here as a string or variable

    ch.check_best(best_model)
```

Step 6: Despite the fact that our models here are more difficult to interpret, there are some ways to get an idea of which features are important. Using the "best model"

```
[ ] features = diabetes.columns[:diabetes.shape[1]]
    importances = random_search.best_estimator_.feature_importances_
    indices = np.argsort(importances)

    plt.title('Feature Importances')
    plt.barh(range(len(indices)), importances[indices], color='b', align='center')
    plt.yticks(range(len(indices)), features[indices])
    plt.xlabel('Relative Importance');
```





In this case study, I looked at predicting diabetes for 768 patients. There was a reasonable amount of class imbalance with just under 35% of patients having diabetes. There were no missing data, and initial looks at the data showed it would be difficult to separate patients with diabetes from those that did not have diabetes.

Three advanced modeling techniques were used to predict whether or not a patient has diabetes. The most successful of these techniques proved to be an AdaBoost Classification technique, which had the following metrics:

Accuracy score for adaboost : 0.7792207792207793

Precision score adaboost : 0.7560975609756098

Recall score adaboost : 0.5636363636363636

F1 score adaboost : 0.6458333333333333

Based on the initial look at the data, it is unsurprising that Glucose, BMI, and Age were important in understanding if a patient has diabetes. These were consistent with more sophisticated approaches. Interesting findings were that pregnancy looked to be correlated when initially looking at the data. However, this was likely due to its large correlation with age.

THANK YOU