Diabetes and Machine Learning via Python

One in 4 dollars spent on healthcare are for diabetes care. Diabetes cost the United States over 176 billion dollars in 2012.

Given this, I was interested in exploring some different machine learning models on a diabetes dataset that I've also posted as well (diab.csv).

Sources: https://care.diabetesjournals.org/content/41/5/929 (https://care.diabetesjournals.org/content/41/5/929)

To run these models, please make sure to install matplotlib, pandas, and numpy. I also recommend using the interface "anaconda" after installing python.

Links are below for more information.

How to actually code python using jupyter notebook: https://jupyter.readthedocs.io/en/latest/install.html (https://jupyter.readthedocs.io/en/latest/install.html)

Installing python/ anaconda on windows: https://problemsolvingwithpython.com/01-Orientation/01.03-Installing-Anaconda-on-Windows/ (https://problemsolvingwithpython.com/01-Orientation/01.03-Installing-Anaconda-on-Windows/)

I recommend googling how to install "pandas" into anaconda and doing the same for "matplotlib" and "numpy"

So much of running a good model is trouble shooting and making sure all libraries (for example, pandas is a library) is installed before / during running the model.

I'm more than happy to provide more details upon request.

In [1]:

```
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
import numpy as np
import seaborn as sns
```

In [2]:

```
diab = pd.read_csv('diab.csv')
print(diab.columns)
diab.head()
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
n',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0.
1	1	85	66	29	0	26.6	0.
2	8	183	64	0	0	23.3	0.
3	1	89	66	23	94	28.1	0.
4	0	137	40	35	168	43.1	2.

In [3]:

```
print("diabetes data dimension: {}". format(diab.shape))
```

```
diabetes data dimension: (768, 9)
```

In [4]:

```
diab.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [5]:

```
print(diab.groupby('Outcome').size())
```

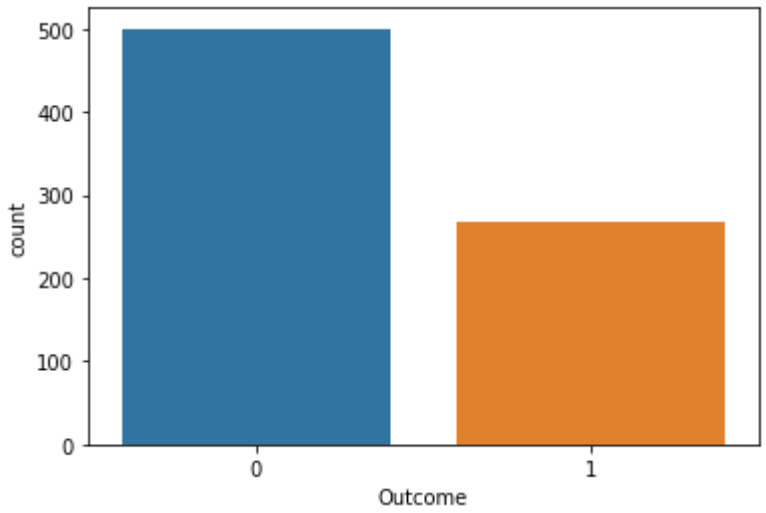
```
Outcome
0    500
1    268
dtype: int64
```

In [6]:

```
sns.countplot(diab['Outcome'],label="Count")
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x19929f57fc8>
```



In [7]:

```
diab.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Main takeaway: This model shows a ~72% test accuracy.

K-NEAREST NEIGHBORS

Most think of this as one of the simplest models to use for predictive analytics.

In this simplicity, there are some decided advantages.

Mainly: No assumptions required No training step required Can be used for both regression and classification Simple and intuitive

Methodology: k-Nearest Neighbors works by literally finding its "nearest neighbors" ... it's a simple algorithm that finds the closest data points in the training datatset. In finding it's nearest neighbor it can make a prediction about a future data point.

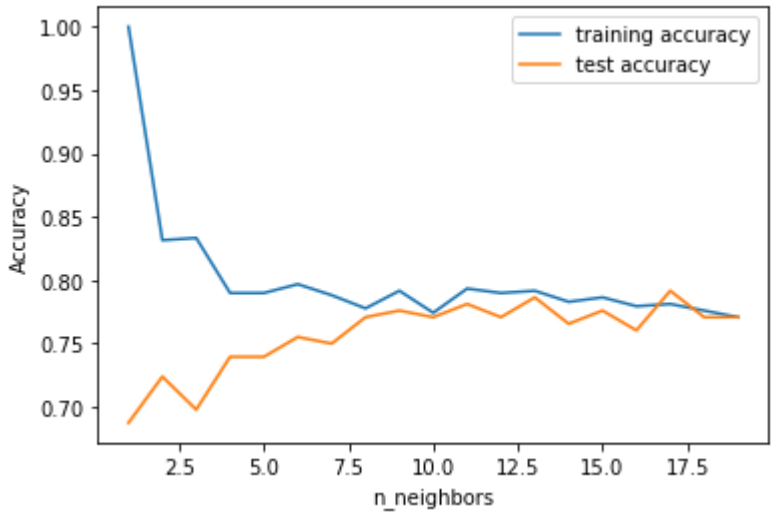
Although this is the simplest model, it takes a bit of time for it to become intuitive and that is okay.

In [8]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(diab.loc[:, diab.columns != 'Outcom
e'], diab['Outcome'], stratify=diab['Outcome'], random_state=66)
```

In [9]:

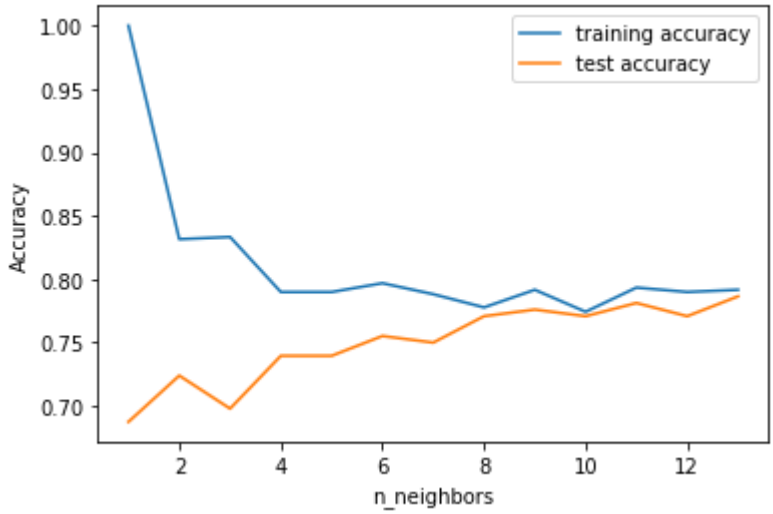
```
from sklearn.neighbors import KNeighborsClassifier

training_accuracy = []
test_accuracy = []
# I start by setting my range... in this case I will try setting the range n_neighbors
 from 1 to 19
# You can always go back and adjust your range
neighbors_settings = range(1, 20)

for n_neighbors in neighbors_settings:
    # first - we will build the model
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(X_train, y_train)
    # secondly - we will record training set accuracy
    training_accuracy.append(knn.score(X_train, y_train))
    # finally - we will record test set accuracy
    test_accuracy.append(knn.score(X_test, y_test))

plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
plt.savefig('knn_compare_model')
```



Based on the above graph we see the nearest neighbor point looks to be somewhere in between 7.5 and 10. To see this graph in more detail, I am going to adjust our training range so that I can find the value between 7.5 and 10 that is most appropriate for our nearest neighbor analysis.

In [10]:

```
from sklearn.neighbors import KNeighborsClassifier

training_accuracy = []
test_accuracy = []
# I start by setting my range... in this case I will try setting the range n_neighbors
 from 2 to 15
# You can always go back and adjust your range
neighbors_settings = range(1, 14)

for n_neighbors in neighbors_settings:
    # first - we will build the model
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(X_train, y_train)
    # secondly - we will record training set accuracy
    training_accuracy.append(knn.score(X_train, y_train))
    # finally - we will record test set accuracy
    test_accuracy.append(knn.score(X_test, y_test))

plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
plt.savefig('knn_compare_model')
```



Based on the above, the best points to use would be somewhere around 9.

In [11]:

```
knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(X_train, y_train)

print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score(X_train, y
_train)))
print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(X_test, y_test
)))
```

```
Accuracy of K-NN classifier on training set: 0.79
Accuracy of K-NN classifier on test set: 0.78
```

LOGISTIC REGREESION

Logistic regression, a classification algorithm, is one of the most common models to use.

Some of its advantages include:

Go to method for binary outcomes. A great example is looking at people with diabetes and those without.

So basically any "yes or no" answer for whether or not someone has a disease... testing logistic regression models can be useful and probably a good place to start.

In [12]:

```
from sklearn import preprocessing
import numpy as np
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression().fit(X_train, y_train)
print("Training set accuracy: {:.3f}".format(logreg.score(X_train, y_train)))
print("Test set accuracy: {:.3f}".format(logreg.score(X_test, y_test)))
```

```
Training set accuracy: 0.786
Test set accuracy: 0.771

C:\Users\Tiffany Taylor\newfolder\lib\site-packages\sklearn\linear_model\_
logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [13]:

```
logreg001 = LogisticRegression(C=0.01).fit(X_train, y_train)
print("Training set accuracy: {:.3f}".format(logreg001.score(X_train, y_train)))
print("Test set accuracy: {:.3f}".format(logreg001.score(X_test, y_test)))
```

```
Training set accuracy: 0.762
Test set accuracy: 0.760

C:\Users\Tiffany Taylor\newfolder\lib\site-packages\sklearn\linear_model\_
logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

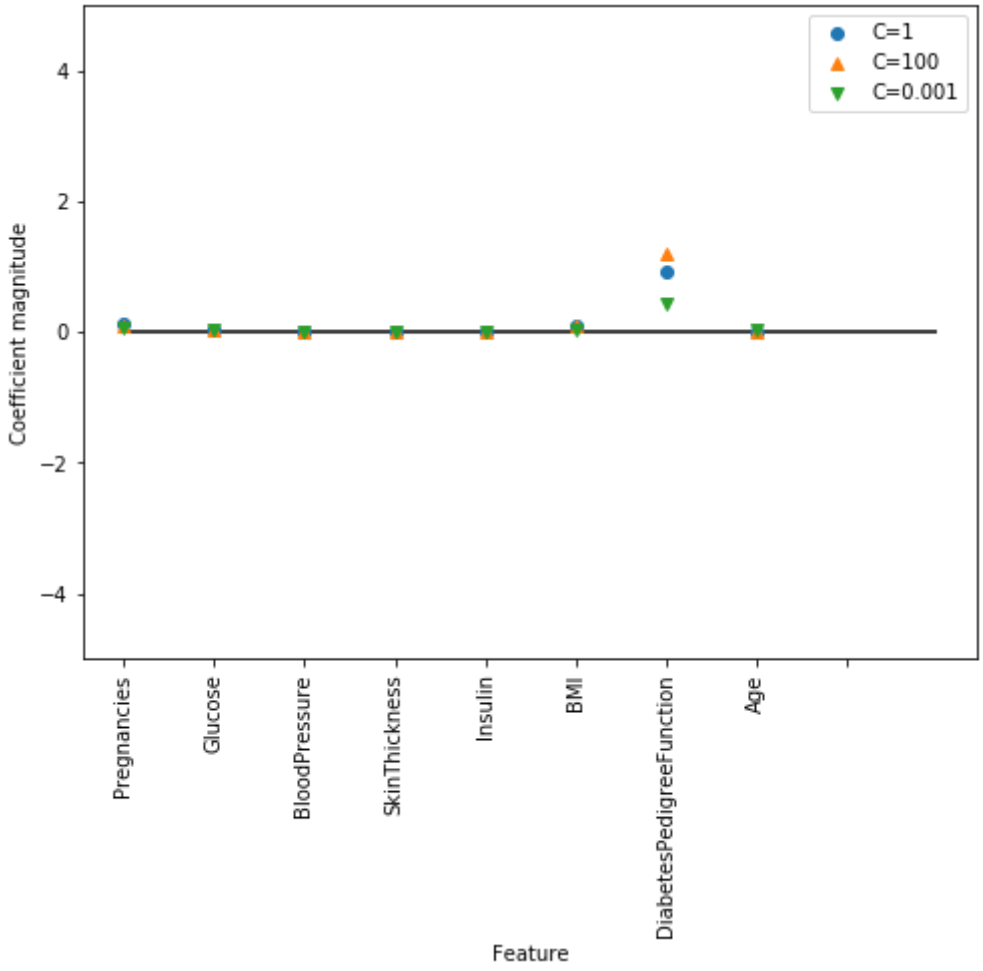
In [14]:

```
logreg100 = LogisticRegression(C=100).fit(X_train, y_train)
print("Training set accuracy: {:.3f}".format(logreg100.score(X_train, y_train)))
print("Test set accuracy: {:.3f}".format(logreg100.score(X_test, y_test)))
```

```
Training set accuracy: 0.783
Test set accuracy: 0.781

C:\Users\Tiffany Taylor\newfolder\lib\site-packages\sklearn\linear_model\_
logistic.py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [15]:

```
diab_features = [x for i,x in enumerate(diab.columns) if i!=8]

plt.figure(figsize=(8,6))
plt.plot(logreg.coef_.T, 'o', label="C=1")
plt.plot(logreg100.coef_.T, '^', label="C=100")
plt.plot(logreg001.coef_.T, 'v', label="C=0.001")
plt.xticks(range(diab.shape[1]), diab_features, rotation=90)
plt.hlines(0, 0, diab.shape[1])
plt.ylim(-5, 5)
plt.xlabel("Feature")
plt.ylabel("Coefficient magnitude")
plt.legend()
plt.savefig('log_coef')
```



DECISION TREE

In [16]:

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.4f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.4f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 1.0000
Accuracy on test set: 0.7135
```

In [17]:

```
tree = DecisionTreeClassifier(max_depth=3, random_state=0)
tree.fit(X_train, y_train)

print("Accuracy on training set: {:.4f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.4f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 0.7726
Accuracy on test set: 0.7396
```

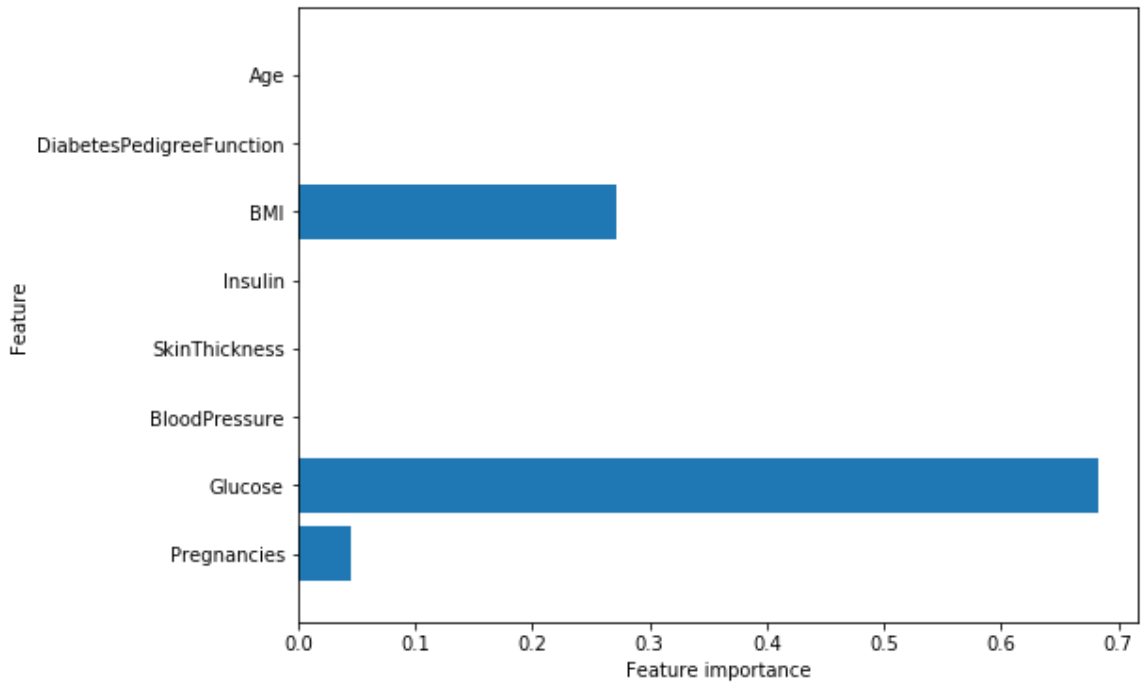
In [18]:

```
print("Feature importances:\n{}".format(tree.feature_importances_))
```

```
Feature importances:
[0.04554275 0.6830362  0.         0.         0.         0.27142106
 0.         0.        ]
```

In [19]:

```
def plot_feature_importances_diab(model):
    plt.figure(figsize=(8,6))
    n_features = 8
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), diab_features)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)

plot_feature_importances_diab(tree)
plt.savefig('feature_importance')
```



SVM: SUPPORT VECTOR MACHINES

I am going to start off this analysis with SVM.

One of SVM's noted advantages is that it works well when there is a clear separtion of classes. It's also a great starting place when working with unfamiliar data sets, especially semi structured data and unstructured data such as text, images, decision trees etc.,

In [20]:

```
from sklearn.svm import SVC

svc = SVC()
svc.fit(X_train, y_train)

print("Accuracy on training set: {:.2f}".format(svc.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(svc.score(X_test, y_test)))
```

```
Accuracy on training set: 0.77
Accuracy on test set: 0.76
```

In [21]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

svc = SVC()
svc.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.2f}".format(svc.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.2f}".format(svc.score(X_test_scaled, y_test)))
```

```
Accuracy on training set: 0.79
Accuracy on test set: 0.80
```

In [22]:

```
svc = SVC(C=1000)
svc.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.3f}".format(
    svc.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(svc.score(X_test_scaled, y_test)))
```

```
Accuracy on training set: 0.944
Accuracy on test set: 0.724
```

In [ ]:

```

```