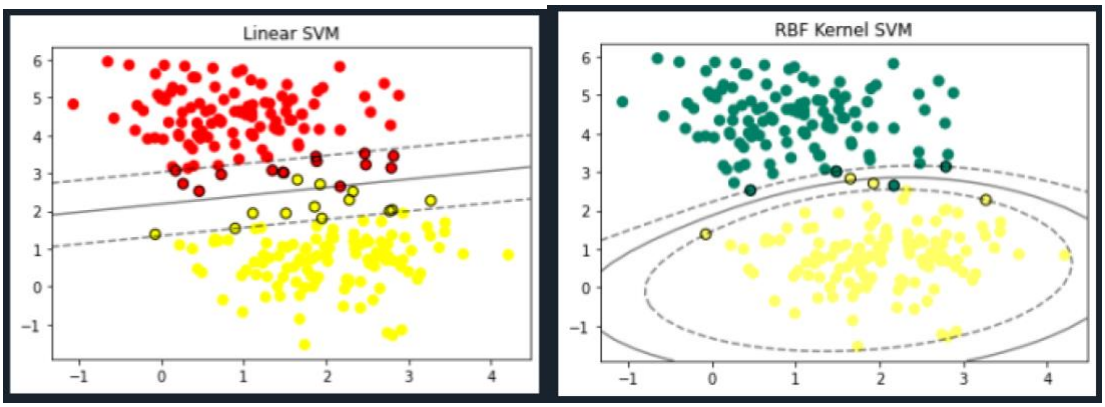
Homework 3
COSC 6342: Machine Learning
University of Houston
Department of Computer Science
Sent on: Sept. 28, 2020; Due: Oct. 6, 2020 (midnight)

Name(s): Garima Singh (1793399)

Ruchi Shah (1800950)

1. Briefly explain the purpose of the kernel function. (15 points)

   The kernel function also referred to as the 'kernel trick' is used to correctly map the non-linearly separable data onto a to a very high dimension or infinity such that the data becomes easily linearly separable. In SVM, the kernel function is also applied on every data point to correctly separate the linearly inseparable data (positive and negative) by maximizing the margin on either side of the hyperplane. As the input dataset is mapped to a higher dimension, the SVM classifier is becomes less sensitive to outliers. This is also one of the reasons why SVM classifier is widely used for classification. When the kernel function is applied to every data point, the original problem remains the same, however the instead of using the input matrix $X \in R^{n*d}$, where n is the number of samples and d is the number of features, we use Kernel matrix $K \in R^{n*n}$ that does not depend of the number of features. This kernel function performs the inner product between two data points using a minimal cost. This calculation is done very efficiently using various available kernel functions like polynomial, radial basis function (RBF), and sigmoid.



The X-Y axes are the input and the output values respectively.
In the above diagram, when we tried to linearly separate the data using a linear SVM and we see that many points on the opposite sides of the decision boundary which means the decision boundary did not accurately separate the data set. However, using an RBF kernel on the same data set, the accuracy to linearly separate the data properly has improved a lot. (The code for the above diagrams is attached as a part of this assignment - GraphsHW3Q1.py)

2. Suppose your dataset has a large number of features. What effect, if any, would feature selection have on an SVM? And what is the effect of raising or lowering the λ hyper-parameter in an SVM? (25 points)

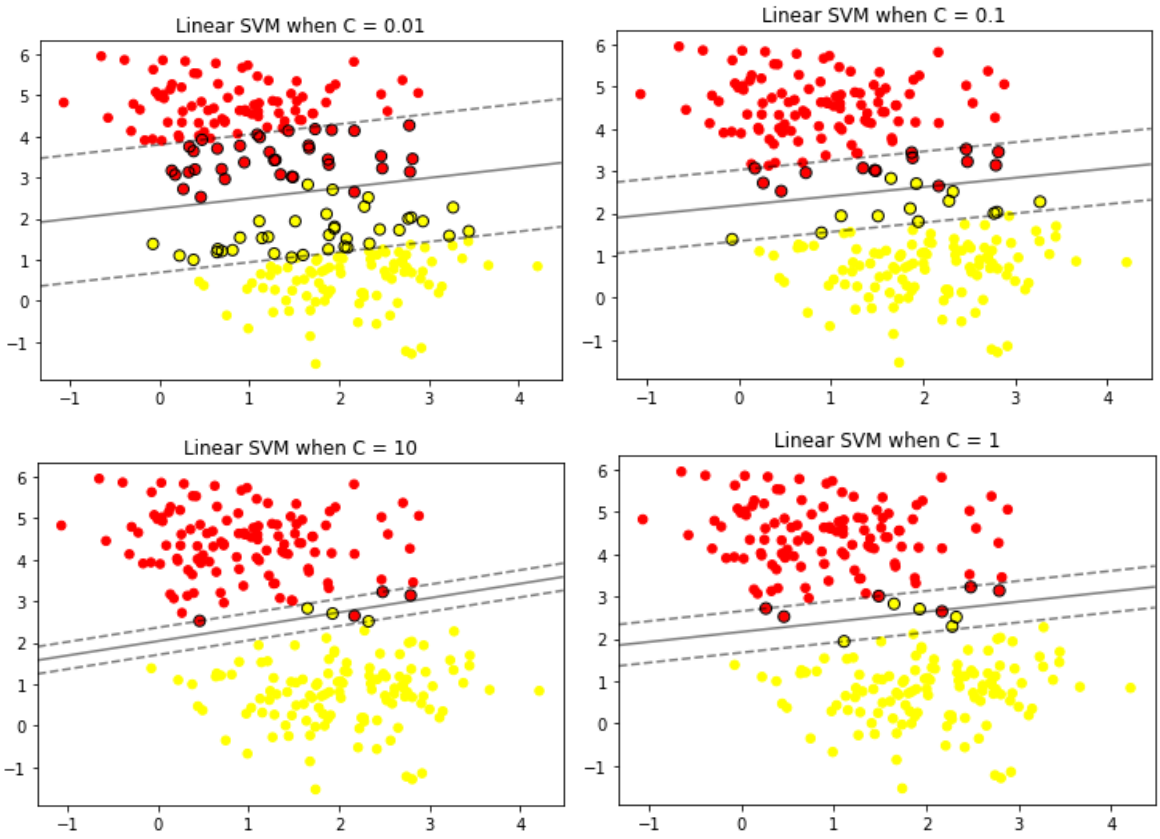
Effect of feature selection:

Linear SVM and all other Kernel SVMs seem provide much better results in higher dimensions. Therefore, SVM are always preferred as a go-to classifier when the number of features are large. The feature selection does not have any direct impact on the accuracy. However, it would help filter out noises from the data set. This benefits the SVM classifier in filtering out the noises from the dataset and reduce the chances of overfitting thereby improving the speed of executing the algorithm.

Effect of λ hyper-parameter:

The equation pertaining to the hyperparameter is below:
$\arg\min_w \; \frac{1}{2}\|w\|_2 \; + \; \lambda \sum e_i$, λ acts as a regularization parameter for SVM and takes in account how much weightage we give to the errors (mis-classified datapoints). Higher the λ, lower (smaller) the

margin. Decreasing the value of λ increases the margin at the cost of accuracy. The graphs below explain this for the case of a linear SVM.



The X-Y axes are the input and the output values respectively.
In the above diagram, we show the effect of lowering and increasing the value of the hyperparameter λ on the decision boundary. When the hyperparameter was the lowest, λ=0.01, there are many points that lie within the margin, thereby affecting the accuracy. The accuracy increases as we increase the hyperparameter, λ=10(The code for the above diagrams is attached as a part of this assignment - GraphsHW3Q2.py)

3. Train an SVM with linear and rbf kernels on the credit dataset, as well as one other kernel of your choice. Optimize the hyper-parameters and settings to try to achieve the best accuracy on the validation set and report your results. Then run your model on the test dataset. Report the accuracy. Graph and attach the classifications on the test set. (60 points):

a. Randomly select 70% of the examples from the data file as training examples, 10% for validation and use the remaining examples for testing. Train the SVM classifier using the sklearn package functionality.
b. Number of Attributes: 24
    i. Attributes include information about: Status of existing checking account, Duration in month, Credit history, Purpose, Credit amount, Savings account/bonds, Present employment, Installment rate in percentage of disposable income, Personal status, Other debtors / guarantors, Present residence, Property, Age, Other installment plans, Housing, Number of existing credits at this bank, Job, Telephone, Foreign worker, etc.
c. Class label: last column in the data file

Methodology followed:

1. Splitting the data in train, validate, test

We have used train_test_split() function from the sklearn library twice in succession to split the data into train, validate and test sets. First, the data was split in to 80:20 ratio as 'model set', 'test set'. The 'model set' was further split into 12.5: 87.5 of 'validate set', train set'. This gives us an overall spilt of the dataset in 70:10:10 into train, validate, test sets respectively.

2. Tuning the hyperparameters

We have tuned the hyperparameters using GridSearchCV() from the sklearn library. GridsearchCV() is an inbuilt function in sklearn's model_selection module wherein we can pass on a string of parameter that we want to vary for the model. The GridSearch returns the model with best scores and also, we can get the corresponding values of the hyperparameter using 'best_params_ '. The hyperparameter used for tuning the model are summarized in the table below and those which gave the best performance on the validation set and used for further testing have been highlighted in red.

Hyperparameter tuning			
Kernel	**C**	**gamma**	**degree**
Linear	0.001, 0.01, 0.04, 0.1, **1**, 10	NA	NA
Rbf (radial basis function)	0.04, 1, 10, 100, **1000**, 10000	0.00001, **0.0001**,0.001,0.01, 0.1	NA
Sigmoid	**0.00001**, 0.001, 0.01	**0.00001**, 0.0001, 0.001, 0.01	NA
Polynomial	100, **1000**, 10000	NA	**2**, 3

3. Model performance

The performance of the model was compared by evaluation the F1 scores and Accuracy and are summarized in the graphs below.

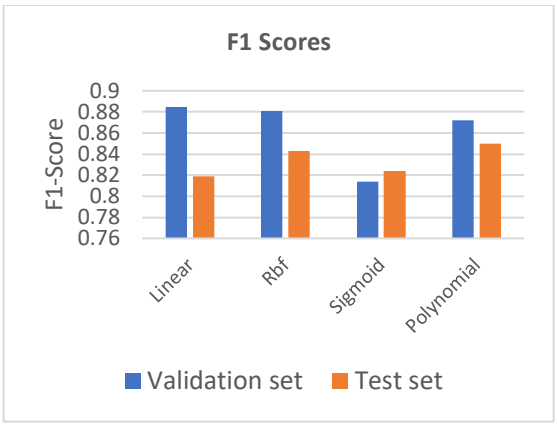
F1 Score as a performance metrics:

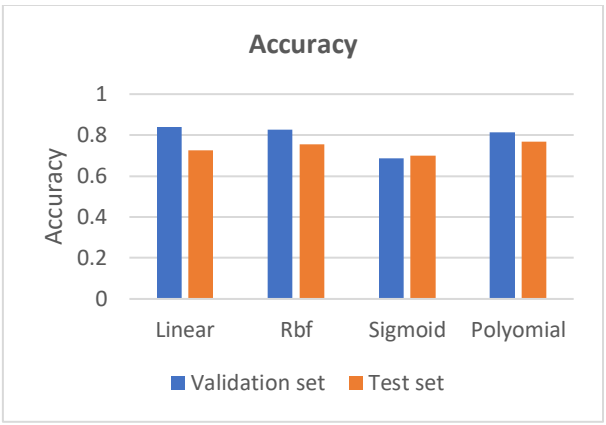
We find that the Polynomial kernel does the best followed by the Rbf kernel on this dataset (we can get an intuition of this from the classification graphs also). Out of the poly and rbf kernel, the polynomial function is more robust an has a comparable performance on the validation and the test sets which is a good indicator.

Accuracy as a performance metrics:

In this case also polynomial and rbf do equally well, and polynomial kernel has a better reproducibility of result on the test set with validation set.

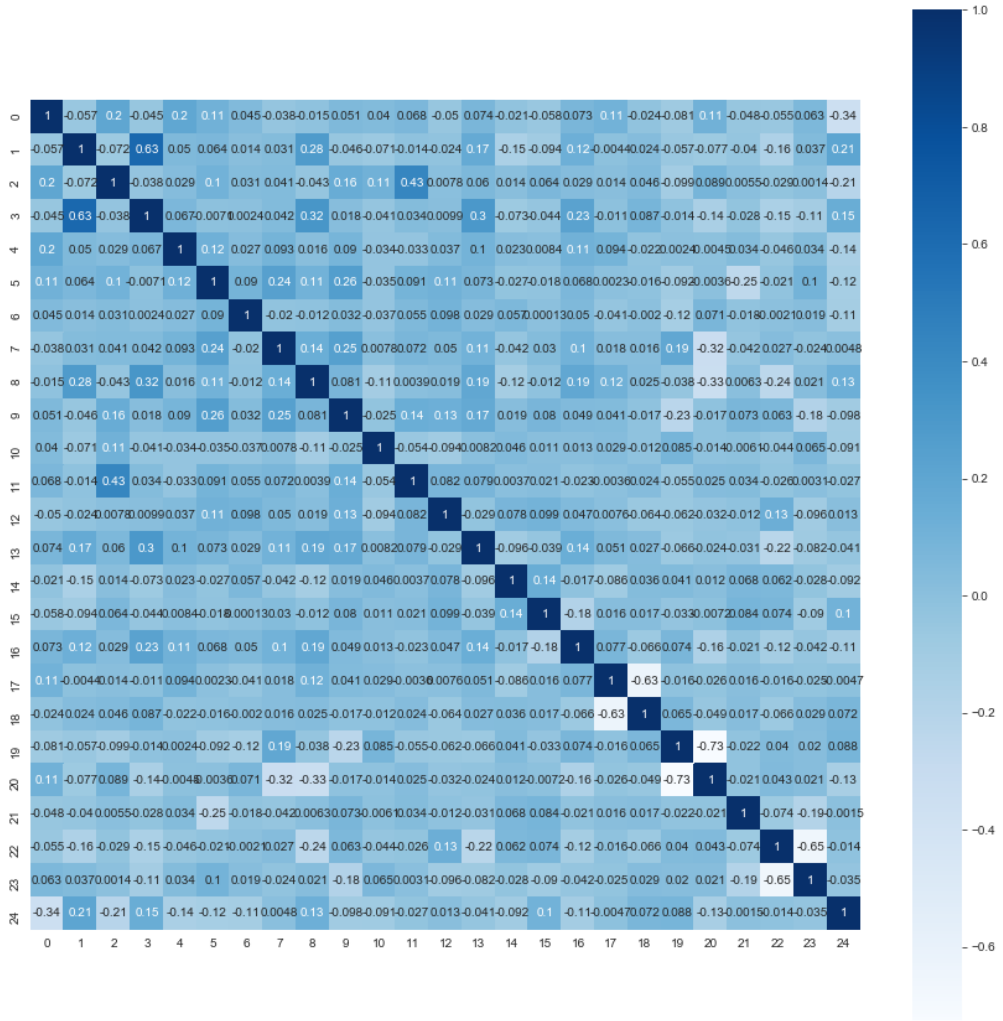
Overall we find that the polynomial kernel performs the best followed by the rbf kernel for this dataset.





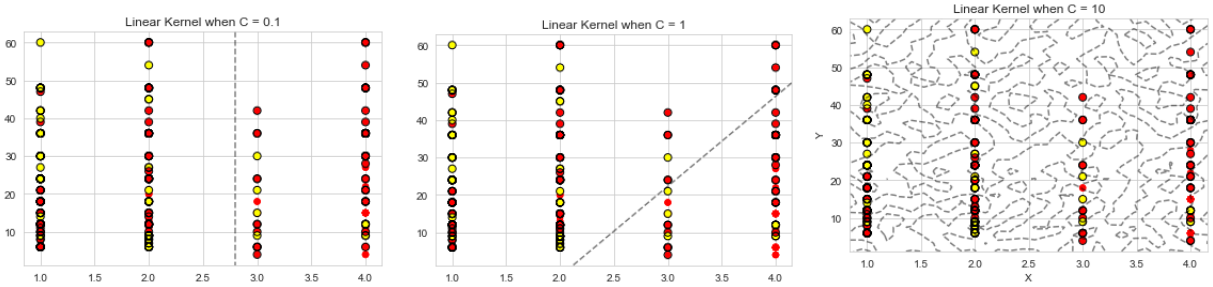
4. Classification graphs:

To select features for plotting, we have used the co-variance matrix as show below. The first and the second features have been selected.

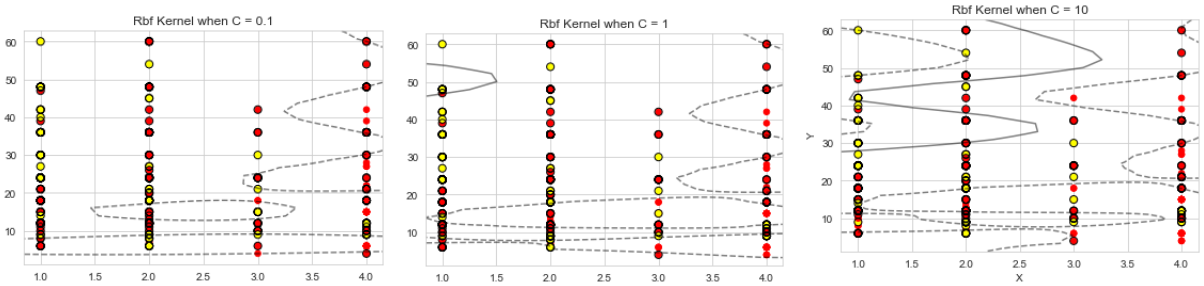


The classification plots for each kernel using two features is as below.

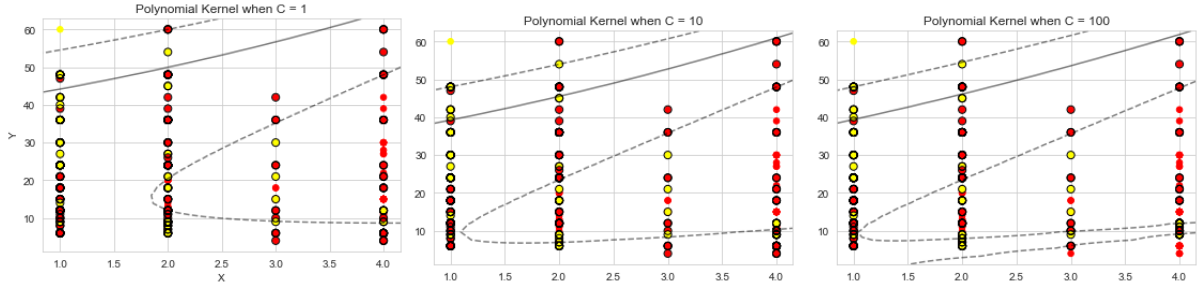
The x-axis is attribute 1 (1st column) and y axis is attribute 2 (2nd column). These are 2D plots with projection (contours) of the hyperplane on the feature's plane.



Here we note that Linear kernel does not have visible a solid decision boundary, all the points are a part of the support vector (dotted soft margin), hence it is not a good classifier as also seen from the performance scores in above chart- this kernel has the worst performance. Also, as we increase the C it quickly leads to overfitting.



For the case of radial basis function, we find solid line for some part of the dataset and for C>1, the higher the C the better the decision line.



The polynomial kernel has a good decision boundary for all values of C with support vectors on both side and or most of the dataset.

Therefore, although this is with just two attributes-it is consistent with the performance metrics. Polynomial is best kernel, followed by rbf and then linear kernel.

Code for hyperparameter tuning of models using all features:

```
#!/usr/bin/env python
# coding: utf-8

# ## Importing libraries

# In[1]:


import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import matplotlib as mlp
import seaborn as sns


# ## Importing dataset

# In[2]:


get_ipython().run_line_magic('matplotlib', 'inline')
sns.set_style('whitegrid')
df = pd.read_csv('SVM_credit_data.txt', delimiter='\s+', header=None)
df.head(5)


# In[3]:


df.describe()


# In[4]:
```

```
# Histograms for the dataset to visualise the data distribution
import matplotlib.pyplot as plt
df.hist(figsize=(15, 15))
plt.show()


# ## Defining attributes and labels

# In[5]:


# Here we define the attributes (X) and the target concept (y)
X=df.iloc[:, 0:-1]
y=df.iloc[:,-1]
print(X)
print(y)


# ## Spilting the dataframe into train, validation and test  set

# In[6]:


# We use the splitting fucntion twice to split the dataset in to train, validate and test sets.
from sklearn.model_selection import train_test_split
X_model, X_test, y_model, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
X_train, X_val, y_train, y_val = train_test_split(X_model, y_model, test_size = 0.125,
random_state = 0)
print(X_train)
print(X_val)
print(X_test)


# ## Model training using different kernels

# In[7]:


from sklearn import svm
from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score


# ### 1. Linear Kernel

# In[8]:


# Model tuning with train and validation set
# Starting point for C as 1/d
parameters = {'C':[0.001, 0.01, 0.04, 0.1, 1, 10]}
svc_linear = svm.SVC(kernel='linear')
clf_linear = GridSearchCV(svc_linear, parameters)
clf_linear.fit(X_train, y_train)
y_pred_val_linear = clf_linear.predict(X_val)

print("The best cross-validated score is:", clf_linear.best_score_)
print("The best estimator is:", clf_linear.best_estimator_)
print("The best param is:", clf_linear.best_params_)
print("metrics for linear kernel on validation set are:")
print(confusion_matrix(y_val, y_pred_val_linear))
print("accuracy score:", accuracy_score(y_val, y_pred_val_linear))
print("f1 score:", f1_score(y_val, y_pred_val_linear))

print(clf_linear.cv_results_)


# In[9]:


# Performance of linear kernel on test set
y_pred_linear = clf_linear.predict(X_test)

print("Metrics for linear kernel on test set are:")
print(confusion_matrix(y_test, y_pred_linear))
print("accuracy score:", accuracy_score(y_test, y_pred_linear))
print("f1 score:", f1_score(y_test, y_pred_linear))


# ### 2. Rbf kernel
```

In[10]:

```
# Model tuning with train and validation set
# Starting point for C as 1/d
parameters = {'C':[0.04, 1, 10, 100, 1000, 10000 ],'gamma':[0.00001, 0.0001,0.001,0.01,
0.1] }
#Based on obeservation scores improves with lower gamma and higher C
svc_rbf = svm.SVC(kernel='rbf')
clf_rbf = GridSearchCV(svc_rbf, parameters)
clf_rbf.fit(X_train, y_train)
y_pred_val_rbf = clf_rbf.predict(X_val)

print("The best cross validation score is:", clf_rbf.best_score_)
print("The best estimator is:", clf_rbf.best_estimator_)
print("The best param is:", clf_rbf.best_params_)
print("The metrics for rbf kernel on validation set are:")
print(confusion_matrix(y_val, y_pred_val_rbf))
print("accuracy score:", accuracy_score(y_val, y_pred_val_rbf))
print("f1 score:", f1_score(y_val, y_pred_val_rbf))
```

In[11]:

```
# Performance of model with rbf kernel on test set
y_pred_rbf = clf_rbf.predict(X_test)

print("The metrics for rbf kernel on test set are:")
print(confusion_matrix(y_test, y_pred_rbf))
print("accuracy score:", accuracy_score(y_test, y_pred_rbf))
print("f1 score:", f1_score(y_test, y_pred_rbf))
```

3. Sigmoidal Kernel

In[12]:

```
# Model tuning with train and validation set
parameters = {'C':[0.00001, 0.001, 0.01],'gamma':[0.00001, 0.0001, 0.001, 0.01] }
svc_sig = svm.SVC(kernel='sigmoid')
```

```
clf_sig = GridSearchCV(svc_sig, parameters)
clf_sig.fit(X_train, y_train)
y_pred_val_sig = clf_sig.predict(X_val)

print("The best corss validation score is:", clf_sig.best_score_)
print("The best estimator is:", clf_sig.best_estimator_)
print("The best param is:", clf_sig.best_params_)
print("The metrics for sigmoid kernel on validation set are:")
print(confusion_matrix(y_val, y_pred_val_sig))
print("accuracy score:", accuracy_score(y_val, y_pred_val_sig))
print("f1 score:", f1_score(y_val, y_pred_val_sig))


# In[13]:


# Performance of model with sigmoid kernel on test set
y_pred_sig = clf_sig.predict(X_test)

print("The metrics for sigmoid kernel are:")
print(confusion_matrix(y_test, y_pred_sig))
print("accuracy score:", accuracy_score(y_test, y_pred_sig))
print("f1 score:", f1_score(y_test, y_pred_sig))


# ### 4. Polynomial Kernel

# In[14]:


# Model tuning with train and validation set
parameters = {'C':[100, 1000, 10000], 'degree':[2, 3]}
svc_poly = svm.SVC(kernel='poly')
clf_poly = GridSearchCV(svc_poly, parameters)
clf_poly.fit(X_train, y_train)
y_pred_val_poly = clf_poly.predict(X_val)

print("The best cross validation score is:", clf_poly.best_score_)
print("The best estimator is:", clf_poly.best_estimator_)
print("The best param is:", clf_poly.best_params_)
print("The metrics for polynomial kernel on validation set are:")
print(confusion_matrix(y_val, y_pred_val_poly))
```

```
print("accuracy score:", accuracy_score(y_val, y_pred_val_poly))
print("f1 score:", f1_score(y_val, y_pred_val_poly))


# In[15]:


# Performance of model with polynomial kernel on test set
y_pred_poly = clf_poly.predict(X_test)

print("The metrics for polynomial kernel on test set are:")
print(confusion_matrix(y_test, y_pred_poly))
print("accuracy score:", accuracy_score(y_test, y_pred_poly))
print("f1 score:", f1_score(y_test, y_pred_poly))


# ## Correlation matrix for feature selection

# In[16]:


# We have used the co-variance matrix to choose the two attributes which have high correlation with the target concept
plt.figure(figsize=(15,15))
df_corr = df.corr()

sns.heatmap(df.corr(), square=True, annot=True, cmap="Blues");
```

Code for graph plots using top two highly correlated feature with the target concept:

```
#!/usr/bin/env python
# coding: utf-8

# ## Importing libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import seaborn as sns


# ## Importing dataset
get_ipython().run_line_magic('matplotlib', 'inline')
sns.set_style('whitegrid')
df = pd.read_csv('SVM_credit_data.txt', delimiter='\s+', header=None)
df.head(5)


# ## Defining attributes and labels

# #### We have selected 1st and 2nd attributes, because they had highest correlation with
the target
X=df.iloc[:,[0,1]]
y=df.iloc[:,-1]


# ## Spilting the dataframe into train test and validation set
from sklearn.model_selection import train_test_split
X_model, X_test, y_model, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
X_train, X_val, y_train, y_val = train_test_split(X_model, y_model, test_size = 0.125,
random_state = 0)


# ## Model training using different kernels
from sklearn import svm
from sklearn.model_selection import cross_val_score
#from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score


# ### 1. Linear Kernel
#Show and plot descision boundary and margin
def show_plot_hyperplane_margin(model, C, title):
    axis = plt.gca()
    xlimit = axis.get_xlim()
    ylimit = axis.get_ylim()
```

```
    # # create a mesh grid to evaluate model
    xmesh = np.linspace(xlimit[0], xlimit[1], 30)
    ymesh = np.linspace(ylimit[0], ylimit[1], 30)
    Ymesh, Xmesh = np.meshgrid(ymesh, xmesh)
    xygrid = np.vstack([Xmesh.ravel(), Ymesh.ravel()]).T
    R = model.decision_function(xygrid).reshape(Xmesh.shape)

    # # Plot both the decision boundary and margins
    axis.contour(Xmesh, Ymesh, R, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-',
'--'])

    # # plot all the support vectors
    axis.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1], s=50,linewidth=1,
facecolors='none', edgecolors='k')
    axis.set_xlim(xlimit)
    axis.set_ylim(ylimit)
    plt.xlabel("X")
    plt.ylabel("Y")
    title = title +' when C = {0}'.format(C)
    axis.set_title(title)
    plt.show();
    return


from matplotlib.colors import ListedColormap
X_set, y_set = X_train.to_numpy(), y_train.to_numpy()
Cparams = [0.1, 1, 10]
for C in Cparams:
    svc_linear = svm.SVC(kernel='linear',C=C)
    svc_linear.fit(X_train, y_train)
    y_pred_linear = svc_linear.predict(X_test)
    print("Linear Kernel C=%d ::", C)
    print("accuracy score:", accuracy_score(y_test, y_pred_linear))
    print("f1 score:", f1_score(y_test, y_pred_linear))
    # ## Plot the decision boundary
    plt.scatter(X_set[:, 0], X_set[:, 1], c=y_set, s=30, cmap='autumn')
    show_plot_hyperplane_margin(svc_linear, C, "Linear Kernel")


# ### 2. Rbf Kernel
print()
Cparams = [0.1, 1, 10]
```

```
#Gparams = [0.01, 1]
for C in Cparams:
    #for G in Gparams:
    svc_rbf = svm.SVC(kernel='rbf',C=C, gamma=0.01)
    svc_rbf.fit(X_train, y_train)
    y_pred_rbf = svc_rbf.predict(X_test)
    print("Rbf Kernel C=%d ::", C)
    print("accuracy score:", accuracy_score(y_test, y_pred_rbf))
    print("f1 score:", f1_score(y_test, y_pred_linear))
    # ## Plot the decision boundary
    plt.scatter(X_set[:, 0], X_set[:, 1], c=y_set, s=30, cmap='autumn')
    show_plot_hyperplane_margin(svc_rbf, C, "Rbf Kernel")


# ### 3. Polynomial kernel
print()
Cparams = [1, 10, 100]
for C in Cparams:
    #for G in Gparams:
    svc_poly = svm.SVC(kernel='poly',C=C )
    svc_poly.fit(X_train, y_train)
    y_pred_poly = svc_poly.predict(X_test)
    print("Poly Kernel C=%d ::", C)
    print("accuracy score:", accuracy_score(y_test, y_pred_poly))
    print("f1 score:", f1_score(y_test, y_pred_poly))
    # ## Plot the decision boundary
    plt.scatter(X_set[:, 0], X_set[:, 1], c=y_set, s=30, cmap='autumn')
    show_plot_hyperplane_margin(svc_poly, C, "Polynomial Kernel")
```