Python as a Calculator

Blank notebook to be used for class exercises.

Exercise 1

Expand the code below to include CountVectorizer parameters in the GridSearchCV with Pipelines. Include the following CountVectorizer Parameters: stop_words = ['english, None], lowercase=[False, True], and min_df=[1, 5, 10]

- What are the parameters that produce the best micro F1?

Best Params: {'clfC': 0.1, 'veclowercase': True, 'vecmin_df': 1, 'vecngram_range': (1, 1), 'vec__stop_words': None}

In [14]:
```
import csv
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import accuracy_score, f1_score
from sklearn.pipeline import Pipeline


import numpy as np
np.random.seed(42)
import random
random.seed(42)

X_txt = []
y = []
with open('./sentiment-twitter-data.tsv') as in_file:
    iCSV = csv.reader(in_file, delimiter='\t')
    for row in iCSV:
        X_txt.append(row[-1])
        y.append(row[-2])


X_txt_train, X_txt_test, y_train, y_test = train_test_split(X_txt, y, test_size=0.2, random_state=42)

pipe = Pipeline([('vec', CountVectorizer()),
                 ('clf', LinearSVC(random_state = 42))])
```

```
params = {'clf__C': [0.01, 0.1, 1.],
          'vec__ngram_range': [(1,1), (1,2)],
          'vec__stop_words': ['english', None],
          'vec__lowercase': [False, True],
          'vec__min_df': [1, 5, 10]}

clf = GridSearchCV(pipe, params, cv = 5, scoring = 'f1_micro')

clf.fit(X_txt_train, y_train)

preds = clf.predict(X_txt_test)

print("Best Score:", clf.best_score_)
print("Best Params:", clf.best_params_)
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to co
nverge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
Best Score: 0.4483688280640125
Best Params: {'clf__C': 0.1, 'vec__lowercase': True, 'vec__min_df': 1, 'vec__ngram_range': (1, 1), 'vec__stop_w
ords': None}
```

Exercise 2

Using the same data as Exercise 1, do the following:

- Apply SelectKBest() with chi2 on the training data. What are the most important features with a k=10?
- Add SelectKBest() to the gridsearch pipeline from Exercise 1. Grid-Search over various choices of K. Choose K on your own.
  - What is the best K? **1000**
  - Did the performance improve over the model trained on the entire set of features? **The model did not improve**

In [15]:
```
from sklearn.feature_selection import SelectKBest, chi2
# WRTIE CODE HERE

vec = CountVectorizer()

X_matrix = vec.fit_transform(X_txt)

skb = SelectKBest(chi2, k = 10).fit(X_matrix, y)

features = np.array(vec.get_feature_names())
```

In [18]:
```
X_txt_train, X_txt_test, y_train, y_test = train_test_split(X_txt, y, test_size=0.2, random_state=42)
```

```python
pipe = Pipeline([('vec', CountVectorizer()),
                 ('skb', SelectKBest()),
                 ('clf', LinearSVC(random_state = 42))])

params = {'clf__C': [0.01, 0.1, 1.],
          'skb__k': [10, 1000, 10000],
          'vec__ngram_range': [(1,1), (1,2)]}

clf = GridSearchCV(pipe, params, cv = 5, scoring = 'f1_micro')

clf.fit(X_txt_train, y_train)

preds = clf.predict(X_txt_test)

print("Best Score:", clf.best_score_)
print("Best Params:", clf.best_params_)
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to co
nverge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to co
nverge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to co
nverge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to co
nverge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to co
nverge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to co
nverge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to co
nverge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to co
nverge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to co
nverge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to co
nverge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/svm/_base.py:976: ConvergenceWarning: Liblinear failed to co
nverge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
Best Score: 0.44649334016393444
Best Params: {'clf__C': 0.01, 'skb__k': 1000, 'vec__ngram_range': (1, 1)}
```

Exercise 3

Apply PCA with 2 dimensions to the iris dataset, then plot the resulting projections.

In [20]:

```
import csv
import numpy as np
X = []
y = []
with open('./iris.csv') as in_file:
    iCSV = csv.reader(in_file, delimiter=',')
    for row in iCSV:
        y.append(row[-1])
        X.append([float(x) for x in row[:-1]])
X = np.array(X)
y = np.array(y)
```

In [23]:

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Write code here

ss = StandardScaler()
X_scale = ss.fit_transform(X)
pca = PCA(n_components = 2)

X_small = pca.fit_transform(X_scale)# Replace this line/variable with the PCA features
```

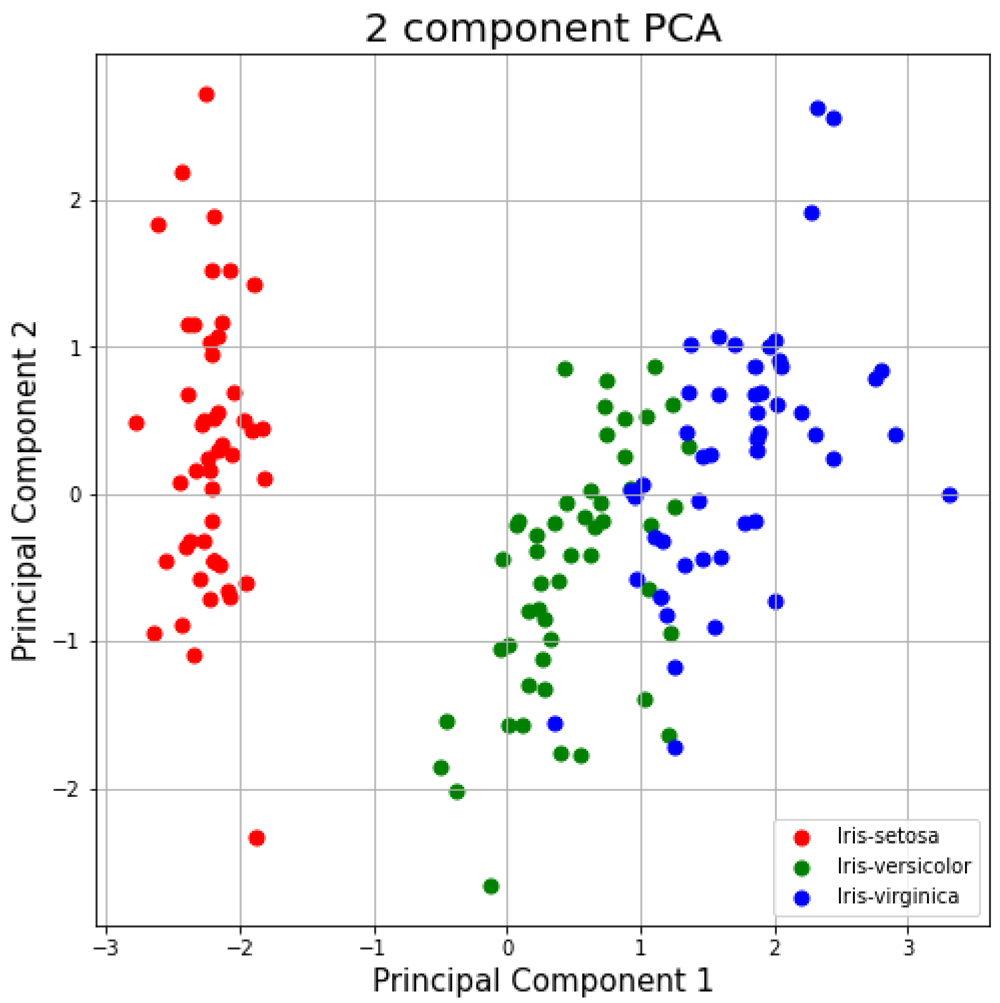
In [24]:

```
%matplotlib inline

# DO NOT Change this cell!!!

import matplotlib
import matplotlib.pyplot as plt

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = y == target
    ax.scatter(X_small[indicesToKeep,0]
               , X_small[indicesToKeep,1]
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



Exercise 4

Using the prima-indians-diabetes.csv'' dataset compare the F1 score of the following missing data strategies:

- Remove all columns with missing values
- Impute using the mean value.
- Impute using the median

What is the best missing data strategy for this dataset?

Note: Missing values are set to 0 in this dataset and only the following columns can have missing values: 2, 3, 4, 5, 6

You will need to convert the 0 values to np.nan.

Here are what the columns represents:

0. Number of times pregnant.
1. Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
2. Diastolic blood pressure (mm Hg).
3. Triceps skinfold thickness (mm).
4. 2-Hour serum insulin (mu U/ml).
5. Body mass index (weight in kg/(height in m)^2).
6. Diabetes pedigree function.
7. Age (years).
8. Class variable (0 or 1).

In [35]:
```
import csv
import numpy as np
X = []
y = []
with open('./prima-indians-diabetes.csv') as in_file:
    iCSV = csv.reader(in_file, delimiter=',')
    for row in iCSV:
        y.append(float(row[-1]))
        X.append([float(x) for x in row[:-1]])
X = np.array(X)
X[X[:,1] == 0,1] = np.nan
X[X[:,2] == 0,2] = np.nan
X[X[:,3] == 0,3] = np.nan
X[X[:,4] == 0,4] = np.nan
X[X[:,5] == 0,5] = np.nan
```

```
print(X[:,3])

y = np.array(y)
```

```
[35. 29. nan 23. 35. nan 32. nan 45. nan nan nan nan 23. 19. nan 47. nan
 38. 30. 41. nan nan 35. 33. 26. nan 15. 19. nan 26. 36. 11. nan 31. 33.
 nan 37. 42. 47. 25. nan 18. 24. nan 39. nan 27. 32. nan 11. 15. 21. 34.
 42. 10. 39. 60. nan 41. nan nan nan 34. nan 27. 30. nan 13. 27. 20. 35.
 nan 20. 30. 20. nan 33. nan 22. 13. nan 26. 28. nan 29. 54. 25. 32. 19.
 nan 15. 40. nan 18. 27. 28. 18. 30. 51. nan nan nan 18. nan 29. nan 28.
 31. 25. 33. 26. 34. nan 32. nan nan nan 23. 15. 56. 39. 30. nan nan 42.
 30. 36. 24. nan 14. nan 37. 31. 13. 20. 26. 25. nan 29. nan 30. 26. nan
 31. 23. 37. 35. nan 17. 50. nan 28. 42. nan 44. 15. 21. 19. 41. 38. 40.
 34. 23. nan 18. 25. nan nan 12. nan 23. 23. 42. 24. 42. nan 46. nan nan
 nan 18. 20. nan nan 28. 36. 41. 39. 35. nan 44. nan nan 20. 41. nan 13.
 44. 27. 16. nan 20. 16. 32. 28. 29. nan 27. 33. 22. 54. 31. 26. 32. 40.
 41. 30. 22. nan 29. nan nan 33. 15. 27. nan 38. 39. 31. nan 37. 25. nan
 28. nan 21. 27. 21. nan 24. 32. nan 22. 35. 15. nan 33. 33. 19. nan nan
 14. 32.  7. 35. 39. 22. 16. 28. 15. nan 32. 15. nan 18. nan 42. nan nan
 37. 32. nan 50. nan 52. 24. 23. nan 10. nan 28. 15. nan nan 26. 44. 39.
 17. 43. 29. 30. 37. 45. nan 31. 38. 29. 25. nan nan 33. 41. nan nan 37.
 23. 14. 19. 28. 30. 37. 17. 10. 31. 22. 11. nan 39. nan 12. 30. 20. 33.
 32. 21. 32. nan 36. 32. 19. 16. nan nan 18. 43. nan nan 34. nan 13. 21.
 35. nan nan 36. 19. nan 19. 32. nan nan 28. 12. nan nan 40. 30. 40. 36.
 33. nan 37. nan 25. 28. nan 17. 16. 28. 48. 23. 22. 40. 43. 43. 15. 37.
 nan 39. 30. 22.  8. 18. 24. 13. 29. 36. 26. 23. 29. nan 14. 12. nan 24.
 34. 40. nan 31. nan nan 41. 25. nan 32. nan nan nan 49. 39. 30. 23. 22.
 35. 33. 21. 32. nan 29. 41. 18. 46. 22. 32. 39. nan 30. 46. 25. nan 16.
 11. nan  8. nan 33. nan 15. nan 25. 23. 27. nan 12. 63. 12. 45. 37. 18.
 13. nan 32. nan 28. 30. nan 28. 48. 33. 22. nan 40. 30. nan 13. 10. 36.
 nan 41. 40. 38. 27. nan nan 27. 45. 17. 38. 31. 30. 37. 22. 31. nan 42.
 41. 32. 17. nan 28. 30. 38. 18. nan nan nan 15. 33. 32. 19. 32. 41. 25.
 39. nan 26. 23. 23. nan 31. 17. nan nan 19. 18. 34. nan nan  7. 32. 33.
 nan nan nan 18. 19. 15. 31. nan 18. nan 52. nan 30. nan nan nan 37. 49.
 40. 25. 32. 23. 29. 35. 27. 21. 43. 31. 28. 30. nan 24. 23. 33. 40. nan
 40. nan nan 32. 34. 19. nan 14. 30. 32. 29. 30. nan nan 31. 17. 30. 47.
 20. nan nan 99. 46. 27. 17. nan 24. 11. nan nan 27. nan 40. 50. nan 22.
 45. 14. nan 19. nan 18. 19. nan 36. 29. nan 32. 42. 25. 39. 13. 21. 22.
 42. 28. 26. nan nan 13. 24. nan 42. 20. nan 27. nan 47. nan nan nan 22.
 nan 40. nan 17. nan nan nan 18. 32. 12. 17. nan nan nan 30. 35. 17. 36.
 35. 25. 25. 23. 40. nan 28. 27. 35. 48. nan 31. nan 43. 46. 46. 39. 45.
 18. 27. 33. 30. 26. 10. 23. 35. nan nan nan nan nan 17. 28. 36. 39. nan
 nan 26. nan 19. 26. 46. nan nan 32. 49. nan 24. 19. nan 11. nan 27. 31.
 29. nan 20. 36. nan 21. nan 32. 13. 27. 36. 20. nan 33. 39. 18. 46. 27.
 19. 36. 29. 30. nan 40. 29. 26. nan nan 23. nan 37. 27. nan 32. 27. 23.
 17. nan 37. 20. 18. nan 37. 33. 41. 41. 22. nan nan 39. 24. 44. 32. 39.
 41. nan nan nan 26. 31. nan 48. 27. 23. nan 31.]
```

In [37]:
```
from sklearn.impute import SimpleImputer
```

```
#from sklearn.experimental import enable_iterative_imputer  # noqa
#from sklearn.impute import IterativeImputer # Uncomment this line to use IterativeImputer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV

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

# IMPUTE WITH MEAN
imp_mean = SimpleImputer(missing_values = np.nan,
                        strategy = 'mean')

X_train_mean = imp_mean.fit_transform(X_train)
X_test_mean = imp_mean.transform(X_test)

clf = SVC()
clf.fit(X_train_mean, y_train)

preds = clf.predict(X_test_mean)

f1 = f1_score(y_test, preds)

print("Mean F1 Value:", f1)

# IMPUTE WITH Median
imp_median = SimpleImputer(missing_values = np.nan,
                          strategy = 'median')

X_train_median = imp_median.fit_transform(X_train)
X_test_median = imp_median.transform(X_test)

clf = SVC()
clf.fit(X_train_median, y_train)

preds = clf.predict(X_test_median)

f1 = f1_score(y_test, preds)

print("Median F1 Value:", f1)

# Remove Missing Columns
mask = np.isnan(X).any(axis = 0)

X_train_rem = X_train[:, ~mask]
X_test_rem = X_test[:, ~mask]
```

```
clf = SVC()

clf.fit(X_train_rem, y_train)

preds = clf.predict(X_test_rem)

f1 = f1_score(y_test, preds)

print("Remove Columns Value:", f1)
```

```
Mean F1 Value: 0.6185567010309277
Median F1 Value: 0.6326530612244898
Remove Columns Value: 0.3564356435643564
```

In [ ]:

```
```