```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import svm, metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

# all individual features
feature_names = ['radius', 'texture', 'perimeter', 'area', 'smoothness', 'compactness', 'concavity', 'concave_points', 'symmetry', 'fractal_dimension']

# either M for malignant or B for benign
target_name = 'Diagnosis'

prediction_data = pd.read_csv("wdbc.csv")

# isolate target variable from features
target = prediction_data[target_name]

# remove the ID from meaningful features
features = prediction_data.drop(['ID', target_name], axis=1)

# create a new dataframe to hold the magnitudes
feature_magnitudes = pd.DataFrame()

# get the combined magnitudes from the different x, y, z and points of each feature's vector
for feature in feature_names:
    # ['radius1', 'radius2', 'radius3'], etc
    feature_dimensions = [f"{feature}{i}" for i in range(1, 4)]

    # calculate and store magnitude in the df
    total = sum(features[dimension]**2 for dimension in feature_dimensions)
    feature_magnitudes[feature] = np.sqrt(total)

# combine target with magnitudes for visualization
target_with_magnitudes = pd.concat([target, feature_magnitudes], axis=1)

# output csv for orange analysis
target_with_magnitudes.to_csv('target_with_magnitudes.csv', index=False)

# only using perimeter/texture features from here because they were the most significant based on Orange analysis of scatter plot
# drop all features except for perimeter and texture
target_perim_texture = target_with_magnitudes.drop([feature for feature in feature_names if feature not in ('perimeter', 'texture')], axis=1)

target = target_perim_texture['Diagnosis']
features = target_perim_texture.drop('Diagnosis', axis=1)

# scale the features using standardscaler
features_scaled = StandardScaler().fit_transform(features)
features = pd.DataFrame(features_scaled, index=features.index, columns=features.columns)
target_perim_texture['perimeter'] = features['perimeter']
target_perim_texture['texture'] = features['texture']

# split data
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.3)

# create and train the linear SVM
clf = svm.SVC(kernel='linear')
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print(f"SVM accuracy: {metrics.accuracy_score(y_test, y_pred)}")

# create the svm line
w = clf.coef_[0]
b = clf.intercept_[0]

# create the X range using a linsapce of the perimeter attributej
svm_x = np.linspace(features['perimeter'].min(), features['perimeter'].max(), 100)

# solve for y using the coef and intercept from the SVM to create y points
svm_y = -(w[0] / w[1]) * svm_x - b / w[1]

# plot the svm line
plt.plot(svm_x, svm_y, c='black', label='SVM Decision Boundary')

# create and train KNN to see if better than SVM
# 2 neighbors because of 2 distinct groups
knn = KNeighborsClassifier(n_neighbors=2)

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# almost always outperformed by SVM
print(F"KNN accuracy: {metrics.accuracy_score(y_test, y_pred)}")

# red for malignant, blue for benign
colors = {
    'M': 'red',
    'B': 'blue'
}

# labels for legend
labels = {
    'M': 'Malignant',
    'B': 'Benign'
}

# scatter the points based on perimeter and texture, color by malignancy
for label in target_perim_texture['Diagnosis'].unique():
    subset = target_perim_texture[target_perim_texture['Diagnosis'] == label]
```

```
    plt.scatter(subset['perimeter'], subset['texture'], color=colors[label], label=labels[label])

# legend and labels
plt.xlabel('Perimeter')
plt.ylabel('Texture')
plt.title('Predicting breast cancer malignancy using a SVM on the perimeter and texture of masses')
plt.legend()

plt.show()
```