In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import warnings
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import RidgeClassifier
from sklearn.metrics import f1_score
warnings.filterwarnings("ignore")
def game_result(x):
    if x['score1'] > x['score2']:
        result = 2
    elif x['score1'] < x['score2']:
        result = 0
    elif x['score1'] == x['score2']:
        result = 1
    return result
```

Objective

The analysis will look at creating a classification adn regression model to predict game outcomes. The classification model will predict game outcomes directly (win, loss or tie) while the regression will predict the score differential between the home and away team.

Dataset

The data set is the Soccer Power Index data from fivethirtyeight. This [link (https://fivethirtyeight.com/methodology/how-our-club-soccer-predictions-work/)](https://fivethirtyeight.com/methodology/how-our-club-soccer-predictions-work/) contains a detailed write up on the SPI model that fivethirtyeight uses as well here is a [link (https://www.espn.com/world-cup/story/_/id/4447078/ce/us/guide-espn-spi-ratings)](https://www.espn.com/world-cup/story/_/id/4447078/ce/us/guide-espn-spi-ratings) to a description written by Nate himself. At a high level the SPI gives a prediction on how probability of winning (0 to 100).

The data set contains following columns (I was unabled to find an offical data dictionary so some of these are my interpretation):

- date - Date of the match
- league_id - An id indicating the league in which the game is being played
- league - The name of the league
- team1 - The home team of the game
- team2 - The away team of the game
- spi1 - The SPI for team 1
- spi2 - The SPI for team 2
- prob1 - The predicted probability of a win for team 1
- prob2 - The predicted probability of a win for team 2
- probtie - The predicted probability of a time between the teams
- proj_score1 - The projected number of goals for team 1
- proj_score2 - The projected number of goals for team 2

- importance1 - An adjustmentment factor applied in the SPI calculation for team 1
- importance2 - An adjustmentment factor applied in the SPI calculation for team 2
- score1 - The number of goals for team 1
- score2 - The number of goals for team 2
- xg1 - The expected goals for team 1 based on their play/chances in the game
- xg2 - The expected goals for team 2 based on their play/chances in the game
- nsxg1 - The non-shot expected goals model for team 1 based on their play/chances in the game
- nsxg2 - The non-shot expected goals model for team 2 based on their play/chances in the game
- adj_score1 - The adjusted score for team 1 based on play
- adj_score2 - The adjusted score for team 2 based on play

Getting & Preparing Data

Data was retried from [here (https://data.fivethirtyeight.com/)](https://data.fivethirtyeight.com/). Below describes the steps in data prepartion

1. Download and unzip data into data folder
2. Use Pandas built in csv reader function
3. Remove games without a score1 and score2 (games that have not yet been played)
4. Create two target variables, one which is a classification problem (did the home team win, lose or tie the game) and another that is a regression problem (the absolute goal difference between home and away team)
5. Create two new feature variables spi_diff and proj_score_diff, these provide the relative difference between the two teams
6. Create a bias term (in this case the bias term will explain the home team advantage)

In [2]:
```
matches = pd.read_csv('data/raw/spi_matches.csv')

# Filter the data set to only matches with a result
matches_played = matches[~(matches['score1'].isnull()) & ~(matches['sco

# Use the game result function to convert the score of a game into a ga
matches_played.loc[:,'class_result'] = matches_played[['score1','score2
class_names = ['loss', 'tie', 'win']
matches_played.loc[:,'reg_result'] = matches_played['score1'] - matches

# New features
matches_played.loc[:,'spi_diff'] = matches_played['spi1']/matches_playe
matches_played.loc[:,'proj_score_diff'] = matches_played['proj_score1']
matches_played.loc[:,'home_team_advnatage'] = 1
```

Create Training, Testing and Validation sets

1. Seperate features variables into X dataframe and the two target variables into their own dataframes (y_class and y_reg)
2. Use SK Learn train_test_split to separate out training from hold out data

3. Use SK Learn to split the hold out data to test and validations

In [3]:
```
# Seperate the dataset into an X and y variable
X = matches_played[['spi1', 'spi2', 'spi_diff', 'prob1', 'prob2', 'prob
y_class = matches_played['class_result']
y_reg = matches_played['reg_result']

#Seperate into a test and train set
## Clasisification splitting
X_train, X_test_val, y_train_c, y_test_val_c = train_test_split(X, y_cl
X_test, X_val, y_test_c, y_val_c = train_test_split(X_test_val, y_test_

## Regressiong splitting
X_train, X_test_val, y_train_r, y_test_val_r = train_test_split(X, y_re
X_test, X_val, y_test_r, y_val_r = train_test_split(X_test_val, y_test_
```

Modeling

A regression and classification model will be made, both will follow the same steps:

1. Create an alpha list to use in the ridge model (alphas will range from 1 to 10,000 increasing by steps of ten)
2. Iterate throught the alpha list training on the train set and finding the accuracy on the test set, loading in the accuracy measure into its list (F1 for classification MSE for regression)
3. Find the alpha that provides the highest accuracy on the testing set and use it in the model, find the accuracy on the validation set to confirm it aligns

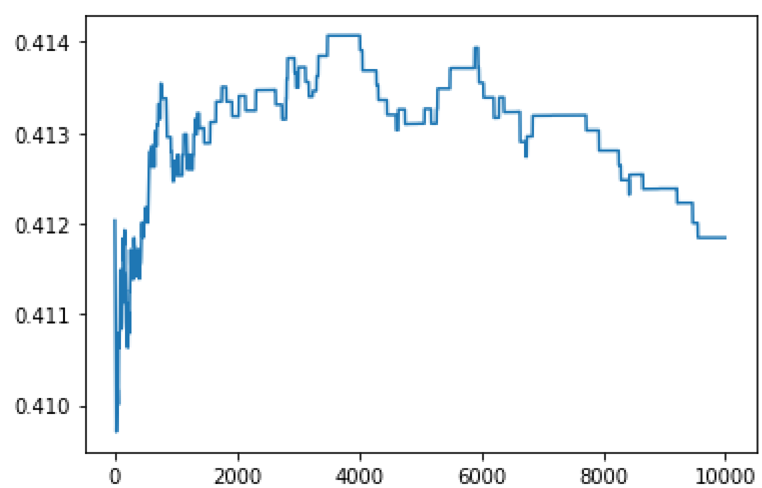
In [13]:
```
alpha_list = list(range(1, 10000, 5))
```

Classification

Increasing alpha improves has the expected impact on the classification mode, it increases accuracy to a point and then shows decreasing returns to scale, it should be noted that overall the change in the F1 score is pretty minimal. The model actually outperforms on the validation set compared to testing.

In [14]:
```python
f1_list = []
for a in alpha_list:
    clf = RidgeClassifier(alpha = a, fit_intercept=False).fit(X_train,
    y_hat = clf.predict(X_test)
    f1 = f1_score(y_test_c, y_hat, average='weighted')
    f1_list.append(f1)

plt.plot(alpha_list, f1_list)
print(alpha_list[f1_list.index(max(f1_list))])

alpha_final_c = alpha_list[f1_list.index(max(f1_list))]
clf_final = RidgeClassifier(alpha = alpha_final_c, fit_intercept=False)
y_hat = clf.predict(X_val)

val_f1 = f1_score(y_val_c, y_hat, average='weighted')
test_f1 = f1_list[f1_list.index(max(f1_list))]
print(f"The validation F1 Score is : {val_f1}")
print(f"The testing F1 Score is: {test_f1}")

```

```
3726
The validation F1 Score is : 0.41594151846626687
The testing F1 Score is: 0.41406964432715754
```



Regression

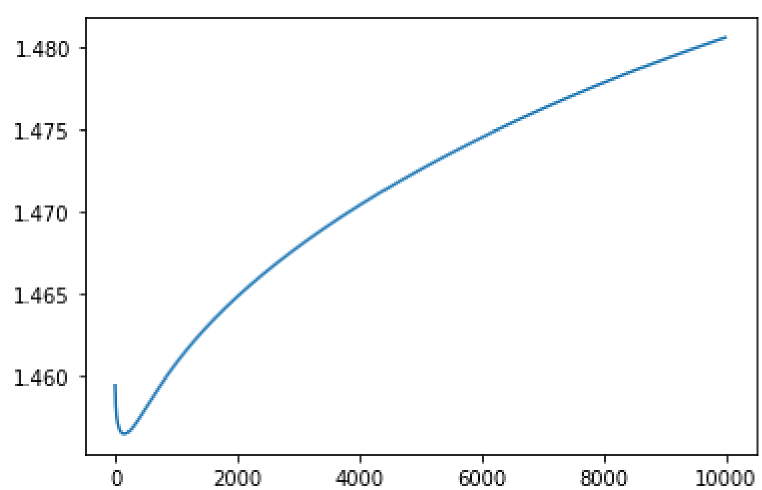
Unlike the classification increasing the alpha has a limited benefit for regression. The MSE errors decreases intitialy by a small amount but then starts increasing. Further the validation set shows a large difference compared to testing.

Additional feature gathering or engineering would be needed.

In [18]:

```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error

r2_list = []
for a in alpha_list:
    reg = linear_model.Ridge(alpha = a, fit_intercept=False).fit(X_trai
    y_hat = reg.predict(X_test)
    r2 = mean_squared_error(y_hat, y_test_r)
    r2_list.append(r2)

plt.plot(alpha_list, r2_list)
print(alpha_list[r2_list.index(min(r2_list))])

alpha_final_r = alpha_list[r2_list.index(max(r2_list))]
reg_final = linear_model.Ridge(alpha = alpha_final_r, fit_intercept=Fal
y_hat = reg_final.predict(X_val)

val_r2 = mean_squared_error(y_hat, y_val_r)
test_r2 = r2_list[r2_list.index(min(r2_list))]
print(f"The validation R2 is : {val_r2}")
print(f"The testing R2 is: {test_r2}")
```

```
151
The validation R2 is : 1.4867415651877667
The testing R2 is: 1.4564326963275747
```



In [ ]:

```
 
```