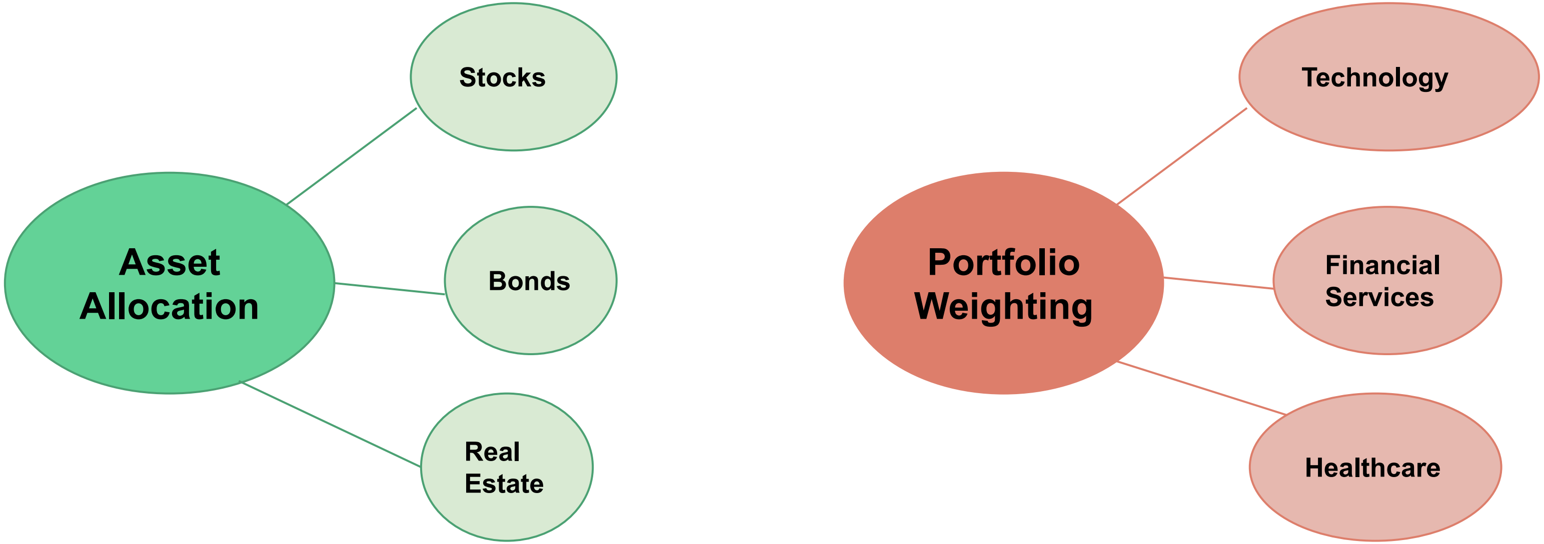
Mutual Funds and SVM

Frankie Contreras, Indra Chaterjee

Project Outline

- **Goal** of this project is be able to predict what mutual funds lead to superior three year returns based on various financial metrics
- The relevant variables ended up being asset allocation and portfolio weighting
- **Assumptions:** A reasonable assumption was made that a mutual fund would follow a similar weighting strategy year after year over the three year time span for average returns



Linear Support Vector Machine (SVM)

- **Hyperplane**

  $\mathbf{w}^T \mathbf{x} + b = 0$

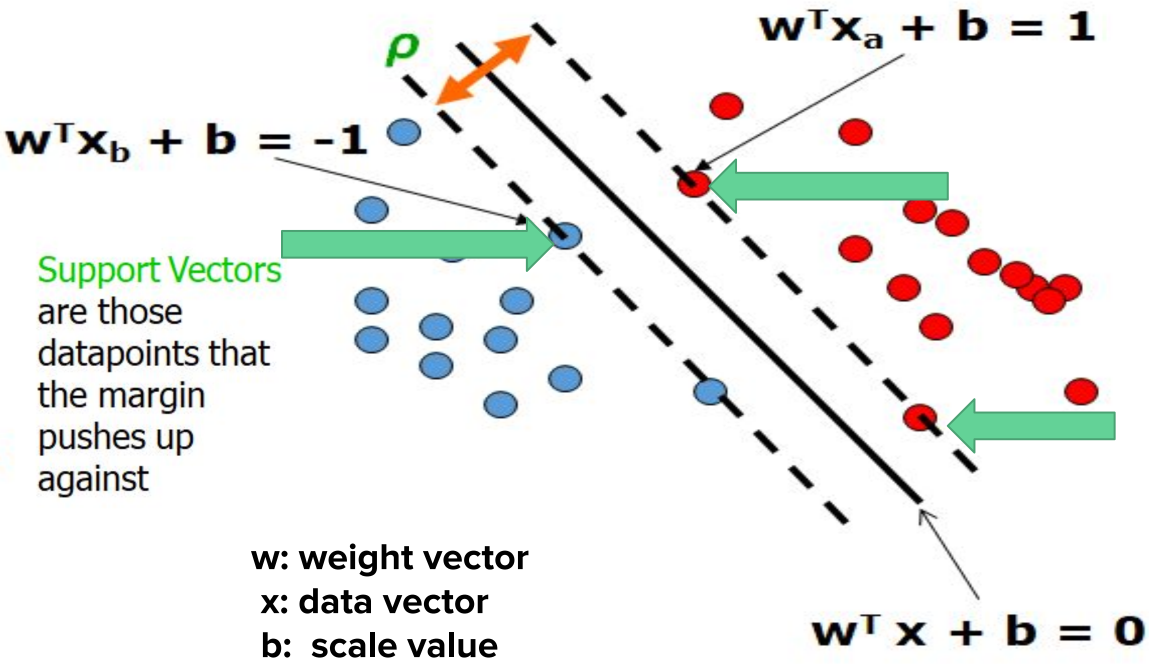
- **Extra scale constraint:**

  $\min_{i=1,\dots,n} |\mathbf{w}^T \mathbf{x}_i + b| = 1$

- This implies:

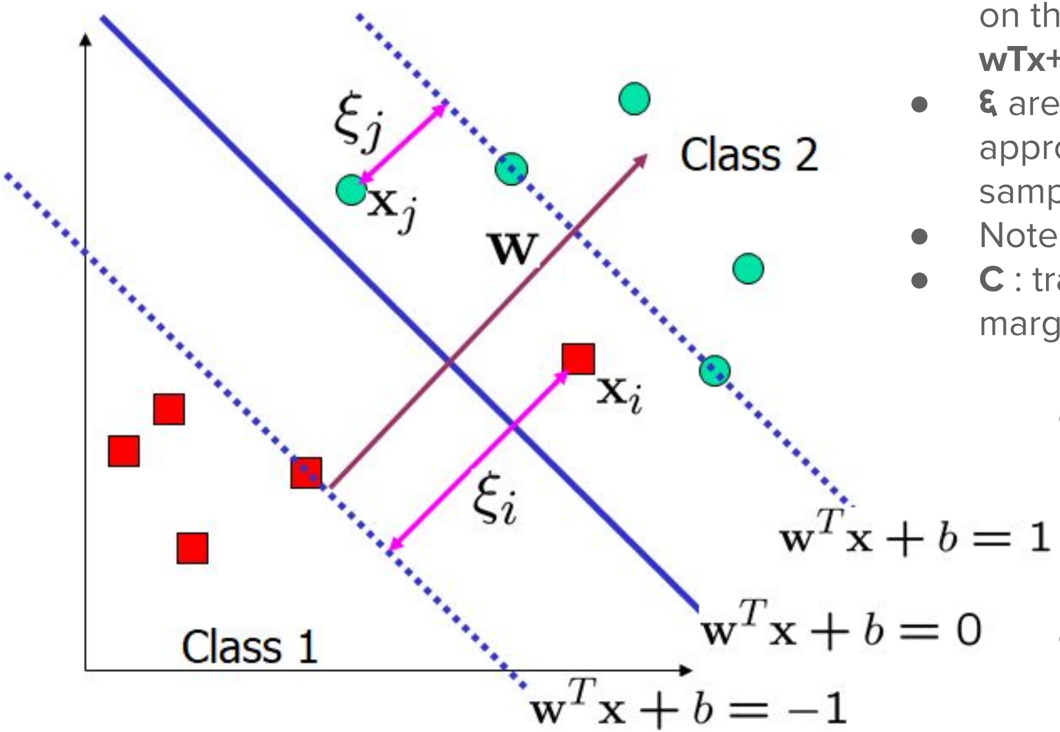
  $w^T(x_a - x_b) = 2$

  $\rho = ||x_a - x_b||_2 = 2/||w||_2$



SVM - What are Slack Variables, C, Gamma ?

Estimate the Margin



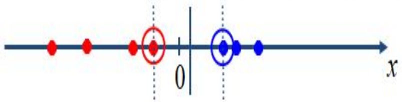
- We allow **ξ“error”** in classification; it is based on the output of the discriminant function **wTx+b**.
- **ξ** are s**lack variables** in optimization that approximates the number of misclassified samples
- Note thatξ=0 if there is no error for xi
- **C** : tradeoff parameter between error and margin

- **gamma** parameter defines how far the influence of a single training example reaches, with low values **meaning** 'far' and high values **meaning** 'close'.
- The **gamma** parameters can be seen as the inverse of the radius of influence of samples selected by the model as **support vectors**

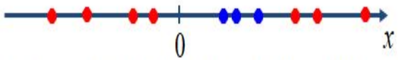
Transforming the Data - Mathematical Convenience

Non-linear SVMs

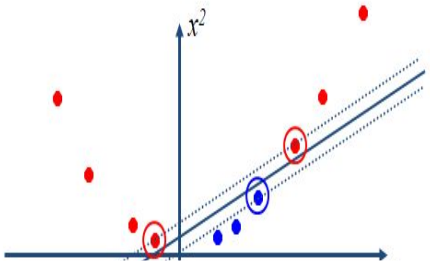
- Datasets that are linearly separable (with some noise) work out great:

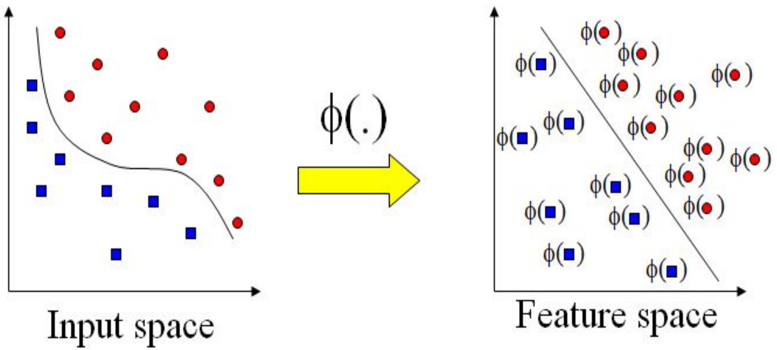


- But what are we going to do if the dataset is just too hard?



- How about ... mapping data to a higher-dimensional space:





Note: feature space is of higher dimension than the input space in practice

Computation in the feature space can be costly because it is high dimensional

- The feature space is typically infinite-dimensional!

The kernel trick comes to rescue

Kernels & The "Kernel Trick"

Why use kernels?

- ❑ Make non-separable problem separable
- ❑ Map data into better representational space

Common kernels

- ❑ Linear
- ❑ Polynomial K(x,z) = (1+xTz)d
  - ❑ Gives feature conjunctions
- ❑ Radial basis function (infinite dimensional space)

- SVM optimization problem

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

- The data points only appear as inner product
- As long as we can calculate the inner product in the feature space, we do not need the mapping explicitly
- Many common geometric operations (angles, distances) can be expressed by inner products
- Define the kernel function K by

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

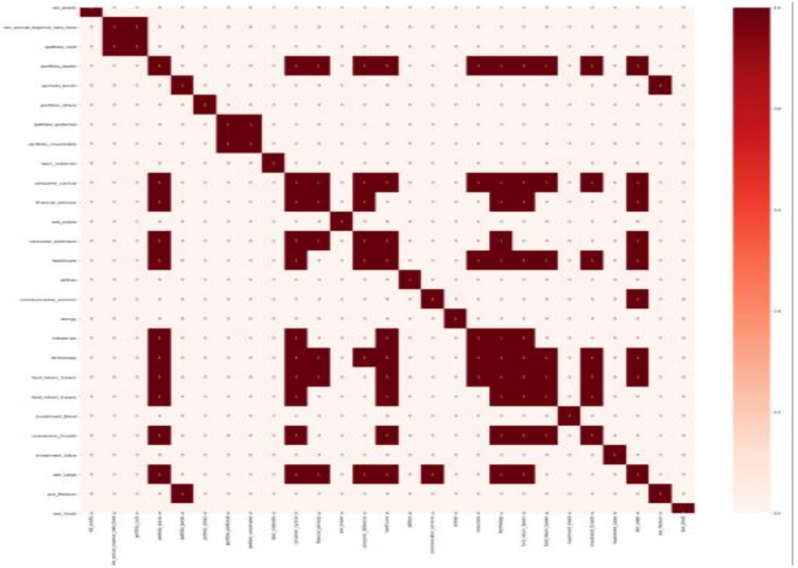
Data - Know the data, feel it, purpose of it

- Data Link: https://www.kaggle.com/stefanoleone992/mutual-funds-and-etfs
- Our data has an interesting investment story, not just numbers:
  - We are trying to build an opportunity by metrics for mutual funds that will lead at least 85% or more accuracy for a higher return on investment
  - 25308 ROWS X 125 Columns
- We domain-expert analyzed and eliminated features that are not relevant to

```
['inception_date', 'currency', 'net_annual_expense_ratio_category', 'price_earnings', 'price_book',
'price_sales', 'price_cashflow', 'bond_maturity', 'bond_duration', 'rating_us_government', 'rating_aa',
'rating_a', 'rating_bbb', 'rating_b', 'rating_below_b', 'rating_others', 'fund_return_ytd',
'category_return_ytd', 'rating_aaa', 'rating_bb', 'fund_return_1month', 'category_return_1month',
 'fund_return_3months', 'category_return_3months', 'fund_return_1year', 'category_return_1year',
'category_return_3years', 'category_return_5years', 'fund_return_10years', 'category_return_10years',
'fund_return_2018', 'category_return_2018', 'fund_return_2017', 'category_return_2017', 'fund_return_2016',
'category_return_2016', 'fund_return_2015', 'category_return_2015', 'fund_return_2014', 'category_return_2014',
'fund_return_2013', 'category_return_2013', 'fund_return_2012', 'category_return_2012', 'fund_return_2011',
'category_return_2011', 'fund_return_2010', 'category_return_2010', 'years_up', 'years_down', 'category_alpha_3ye
'category_alpha_5years', 'fund_alpha_10years', 'category_alpha_10years', 'category_beta_3years',
'category_beta_5years', 'fund_beta_10years','category_beta_10years', 'fund_mean_annual_return_3years',
'category_mean_annual_return_3years', 'fund_mean_annual_return_5years', 'category_mean_annual_return_5years',
'fund_mean_annual_return_10years', 'category_mean_annual_return_10years','category_r_squared_3years',
'category_r_squared_5years', 'fund_r_squared_10years', 'category_r_squared_10years', 'category_standard_deviatio
'category_standard_deviation_5years', 'fund_standard_deviation_10years', 'category_standard_deviation_10years',
'category_sharpe_ratio_3years', 'category_sharpe_ratio_5years', 'fund_sharpe_ratio_10years', 'category_sharpe_rat
'category_treynor_ratio_3years', 'category_treynor_ratio_5years',  'fund_treynor_ratio_10years', 'category_treyno
```

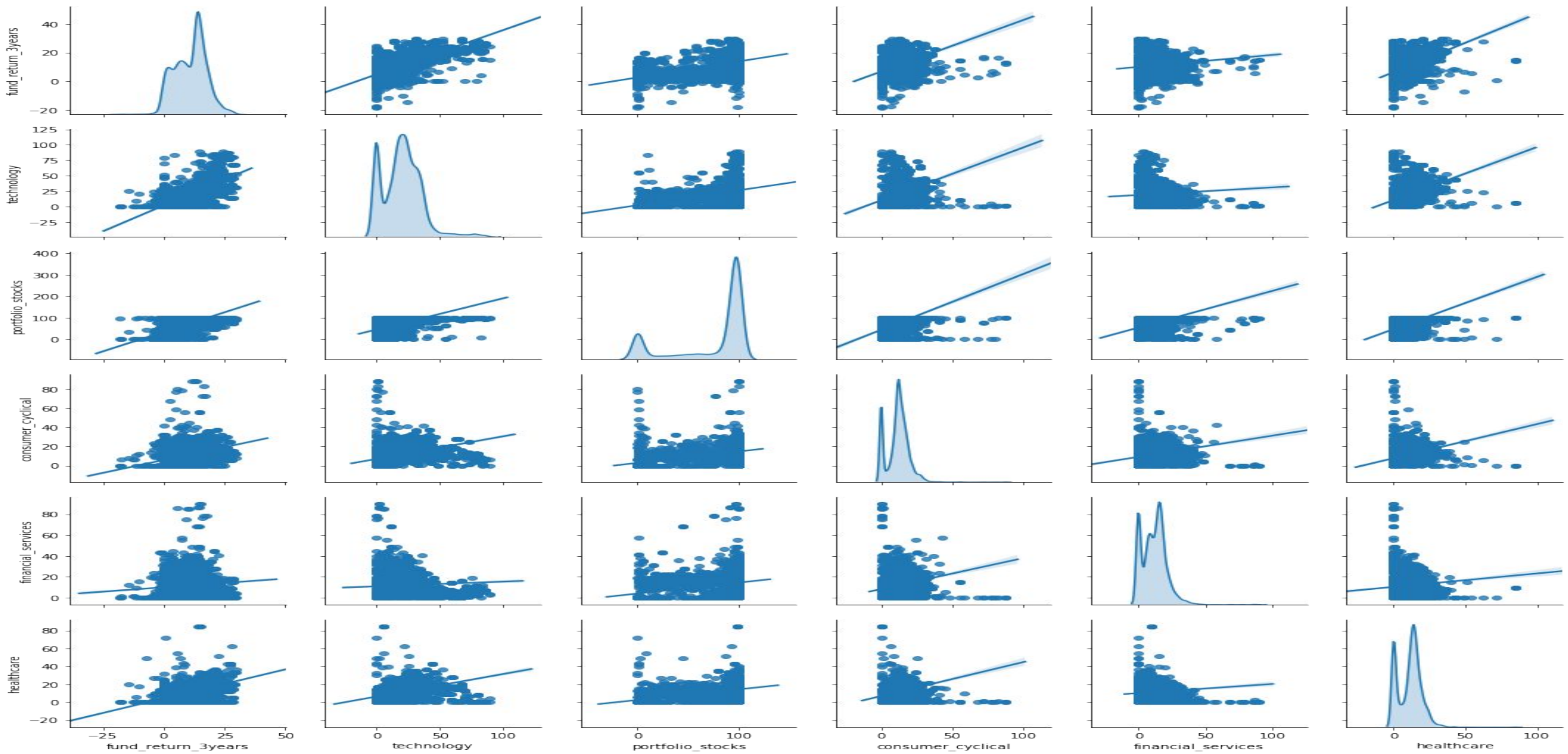
Data Cleaning

- We read the data , visualized and got rid of NANs (insignificantly low%)
  - Identified ‘Investment’ ('Blend', 'Growth', 'Value') and ‘Size’ ('Large', 'Medium', 'Small') as candidates for transformation
  - Transformed Investment Style and Size into “dummies” with 0 or 1
  - We will not use categoricals in SVM but still interesting to see for co-relations



- Top Variables - For 3 year returns we see high correlations with tech, investment growth, healthcare, consumer cyclical, etc

Data Distribution



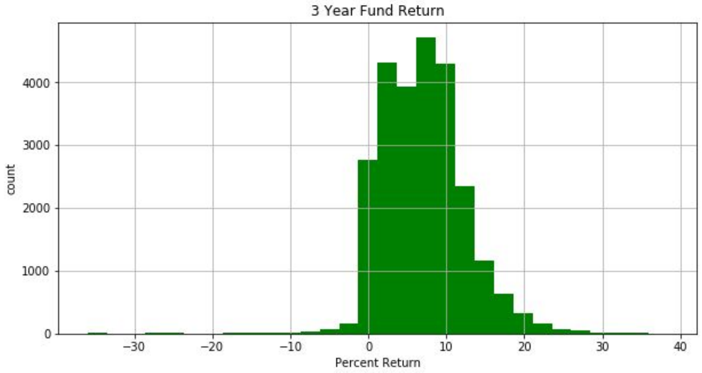
Feature Engineering

- ❑ Made a dataframe for 3 Year Fund Return
- ❑ Converted 3 year Return to Binary
  - ❑ 1 = Superior Fund
  - ❑ 0 = Non-Superior Fund
- ❑ Need to make sure we have even distribution of both superior & Non-Superior Return
- ❑ S&P 500 avg came 7.57%
- ❑ To be more profitable than the market (after fees) we need a fund that returns >=13.12% (7% avg + %1.12 + 3%(assume low load fee) → **say, 13%**

- ❑ Need to make the number classifications (1's = superior and 0's = non-superior) as evenly as possible, so we will have to pick at random less than <13% and number of data points equal to the number of data points above >=13%

In [20]:
```
df3.fund_return_3years.hist(figsize=(10,5), color='g', bins=30)
plt.title('3 Year Fund Return')
plt.xlabel('Percent Return')
plt.ylabel('count')

print("3 years fund return is", df3.fund_return_3years.mean())
```

```
3 years fund return is 6.996638878964687
```



In [75]:
```
df3['return_binary'] = df3['fund_return_3years'].apply(lambda x: 1 if x >= 13 else 0)
```

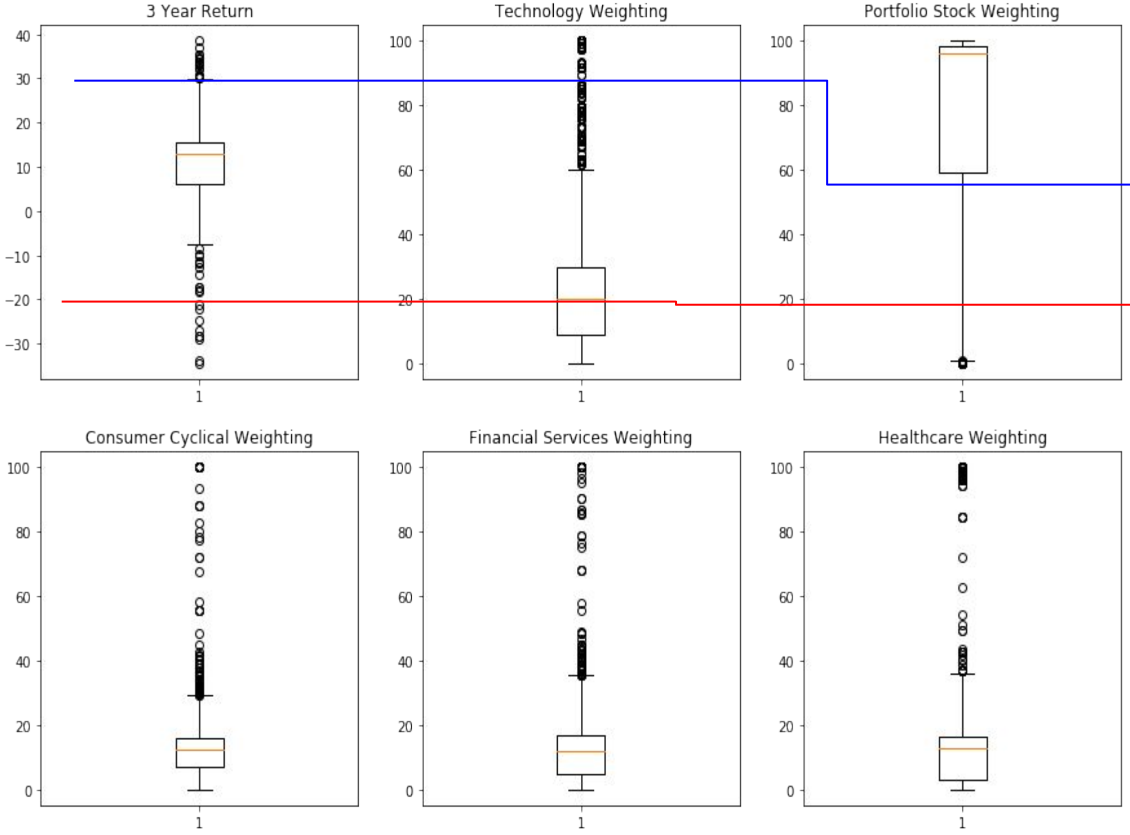
In [76]:
```
df3_0 = df3[df3.return_binary == 0]
df3_1 = df3[df3.return_binary == 1]
print(len(df3_0))
print(len(df3_1))
```

```
22296
3012
```

In [78]:
```
print("We sample df3_0 at 13.5 percent to get %d random rows" %(len(df3_0.sample(frac=0.135))))
df3_0_sub = df3_0.sample(frac=0.135)
```

```
We sample df3_0 at 13.5 percent to get 3010 random rows
```

Outliers



- Made Fund returns more realistic <30% and >-25%
- It would also make sense that an investor may want to invest in a mutual fund that is at least somewhat diversified, even if it is just a little. Thus we dropped funds that are over 90% weighting in an individual sector

Scaling

- Initially scaled to bring all ranges to from 0 to 1, however may not be necessary when categoricals get dropped

	fund_return_3years	technology	portfolio_stocks	investment_Growth	consumer_cyclical	financial_services	size_Large	healthcare	return_binary
mean	11.045292	20.089397	74.378126	0.503585	11.491931	11.650722	0.600205	11.124222	0.496927
max	29.710000	89.630000	100.000000	1.000000	87.870000	90.000000	1.000000	84.560000	1.000000
min	-15.810000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

In [91]:
```
import sklearn
from sklearn import preprocessing
df3_scale = preprocessing.MinMaxScaler().fit_transform(df3)
```

	consumer_cyclical	technology	portfolio_stocks	financial_services	healthcare
0	0.273234	0.9760	0.197792	0.090222	0.194063
1	0.347763	0.9894	0.188233	0.069333	0.195482
2	0.401540	0.9876	0.193923	0.106111	0.168992
3	0.777418	0.9849	0.111528	0.018778	0.146641
4	0.155194	0.9677	0.046660	0.264667	0.161306

SVM Results

- Ran SVM on default parameters: Kernel = Linear, C = 1 on a (70/30 split)

SVM (Support Vector Machine)

```
In [100]:  from sklearn import svm
           from sklearn.svm import SVC

In [101]:  clf = svm.SVC(kernel = 'linear')
           clf.fit(X_train, Y_train)

Out[101]:  SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
               decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
               kernel='linear', max_iter=-1, probability=False, random_state=None,
               shrinking=True, tol=0.001, verbose=False)

In [102]:  y_pred = clf.predict(X_test)

In [102]:  y_pred = clf.predict(X_test)

In [103]:  from sklearn import metrics
           print("Metrics for SVM 3 Year Returns model are:")
           print()
           print("Accuracy:", metrics.accuracy_score(Y_test, y_pred))
           print("Precision:", metrics.precision_score(Y_test, y_pred))
           print("Recall:", metrics.recall_score(Y_test, y_pred))
           print('AUC score is:', metrics.roc_auc_score(Y_test, y_pred))

           Metrics for SVM 3 Year Returns model are:

           Accuracy: 0.8589306029579067
           Precision: 0.8597914252607184
           Recall: 0.85385500575374
           AUC score is: 0.8588735096260263
```

Metrics show to be pretty high at ~85%

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

**Note AUC score is the area under the ROC curve, which is the TP Rate vs FP Rate. Higher AUC means higher probabilities from the positive classes separated from the negative classes*

SVM Parameter Tuning

- Used C Grid search for hyperparameter tuning, which will pick best parameters from a specified range
- Best parameters are determined by highest Recall Score
- Overall Performance is better after tuning ~88%

```
In [175]: from sklearn.model_selection import GridSearchCV
          parameters = {'kernel':('linear', 'rbf'), 'C':[0.1, 0.5, 1, 2, 3, 4, 5, 10],
                        'gamma':[1e-3, 1e-4, 1e-5, 0.01, 0.1, 1, 2, 3, 10]}

In [176]: from sklearn import svm
          from sklearn.svm import SVC
          sv5 = svm.SVC()

In [180]: clfsv = GridSearchCV(sv5, parameters)
          clfsv.fit(X_train, Y_train)

Out[180]: GridSearchCV(cv='warn', error_score='raise-deprecating',
                       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
                                     decision_function_shape='ovr', degree=3,
                                     gamma='auto_deprecated', kernel='rbf', max_iter=-1,
                                     probability=False, random_state=None, shrinking=True,
                                     tol=0.001, verbose=False),
                       iid='warn', n_jobs=None,
                       param_grid={'C': [0.1, 0.5, 1, 2, 3, 4, 5, 10],
                                   'gamma': [0.001, 0.0001, 1e-05, 0.01, 0.1, 1, 2, 3,
                                             10],
                                   'kernel': ('linear', 'rbf')},
                       pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                       scoring=None, verbose=0)

In [181]: print(clfsv.best_params_)

          {'C': 10, 'gamma': 3, 'kernel': 'rbf'}
```

```
sv5 = svm.SVC(kernel='rbf', C = 10, gamma = 3)
sv5.fit(X_train, Y_train)
y_pred = sv5.predict(X_test)
print("Metrics for C = 10, kernel = rbf, gamma = 3")
print()
print("Accuracy:", metrics.accuracy_score(Y_test, y_pred))
print("Precision:", metrics.precision_score(Y_test, y_pred))
print("Recall:",metrics.recall_score(Y_test, y_pred))
print('AUC score is:', metrics.roc_auc_score(Y_test, y_pred))
```

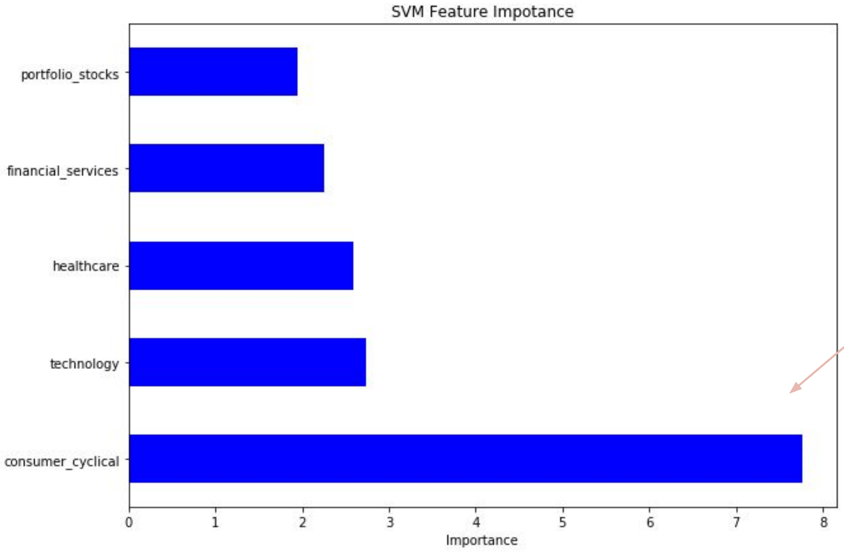
```
Metrics for C = 10, kernel = rbf, gamma = 3

Accuracy: 0.8805460750853242
Precision: 0.8731596828992072
Recall: 0.8872266973532796
AUC score is: 0.8806212226923879
```

	Model 1	Model 2 (C Grid)
Accuracy	85.9%	88.1%
Precision	85.9%	87.3%
Recall	85.4%	88.7%
AUC Score	85.6%	88.1%

Two Feature Results

- Importance of a feature is calculated by calculating the increase in the model's prediction error after permuting the feature. A feature is "important" if shuffling its values increases the model error, because in this case the model relied on the feature for the prediction
- Will pick the top 2 to model **(consumer cyclical and technology)**

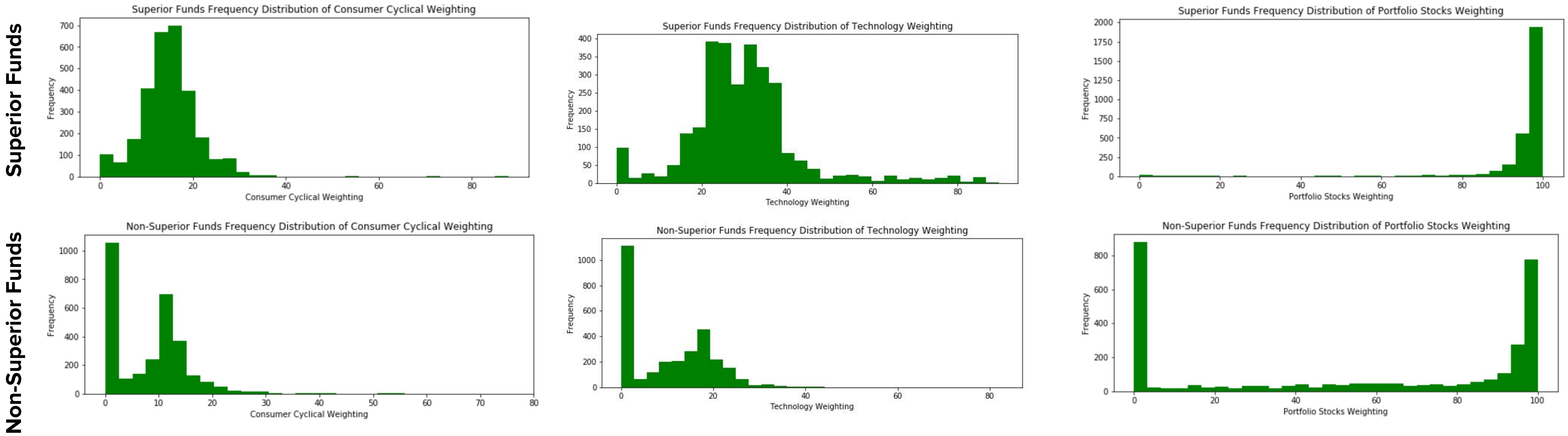
```
In [63]:  pd.Series(abs(clf.coef_[0]), index=df3_scale_df.columns).nlargest(10).plot(kind='barh', figsize=(10,7), color='b')
          plt.title('SVM Feature Impotance')
          plt.xlabel('Importance')

Out[63]:  Text(0.5, 0, 'Importance')
```



Consumer cyclical weighting is nearly 2.5x more impactful than every other feature

Mutual Fund Distribution

- Consumer cyclical weighting has a center distribution ~18% for superior funds while non-superior had many 0 weightings and has a distribution centered around 12%
- Similar distribution seen for technology



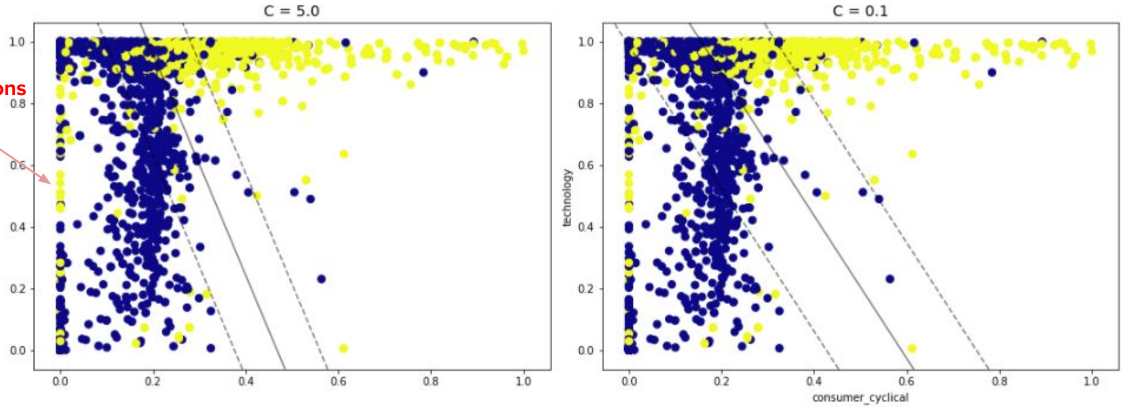
Model Comparison

- Standard linear kernel with C = 5 and C = 0.1

```
In [190]:
#code privded by https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html
fig, ax = plt.subplots(1, 2, figsize=(16, 6))
fig.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)

for axi, C in zip(ax, [5.0, 0.1]):
    model = SVC(kernel='linear', C=C).fit(x_train, y_train)
    axi.scatter(x_train[:, 0], x_train[:, 1], c=y_train, s=50, cmap='plasma')
    plot_svc_decision_function(model, axi)
    axi.scatter(model.support_vectors_[:, 0],
                model.support_vectors_[:, 1],
                s=300, lw=1, facecolors='none');
    axi.set_title('C = {0:.1f}'.format(C), size=14)
plt.ylabel('technology')
plt.xlabel('consumer_cyclical')

Out[190]: Text(0.5, 0, 'consumer_cyclical')
```

```
Metrics for C = 5

Accuracy: 0.863481228668942
Precision: 0.8482834994462901
Recall: 0.8814729574223246
AUC score is: 0.8636836103197112
```

Some misclassifications

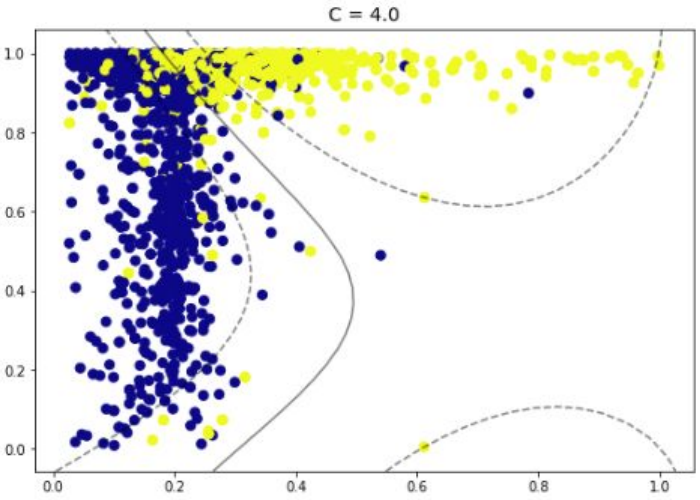



```
Metrics for C = 0.1

Accuracy: 0.8577929465301479
Precision: 0.8207253886010363
Recall: 0.9113924050632911
AUC score is: 0.8583958650738278
```

Two Feature C Grid Search & Model Comparison

- C Grid increased some metrics but not a big improvement

```
In [199]: print(clf.best_params_)

{'C': 4, 'gamma': 2, 'kernel': 'rbf'}
```




```
Metrics for C = 4, kernel = rbf, and gamma = 2

Accuracy: 0.8518253400143164
Precision: 0.8553530751708428
Recall: 0.9037304452466908
AUC score is: 0.8396744099024973
```

After comparing all the models, Model 2 *(5 feature C grid tuned)* seems to have overall higher metrics

	Model 1	**Model 2** (c grid)	**Model 3** (2 features, c = 5)	**Model 4** (2 features, c = 0.1)	**Model 5** (c grid, 2 features)
Accuracy	85.9%	88.1%	86.3%	85.8%	85.2%
Precision	85.9%	87.3%	84.8%	82.1%	85.5%
Recall	85.4%	88.7%	88.1%	91.1%	90.4%
AUC Score	85.6%	88.1%	86.4%	85.8%	83.9%

Pyspark

- Followed same procedure as scikit

Variable Selection

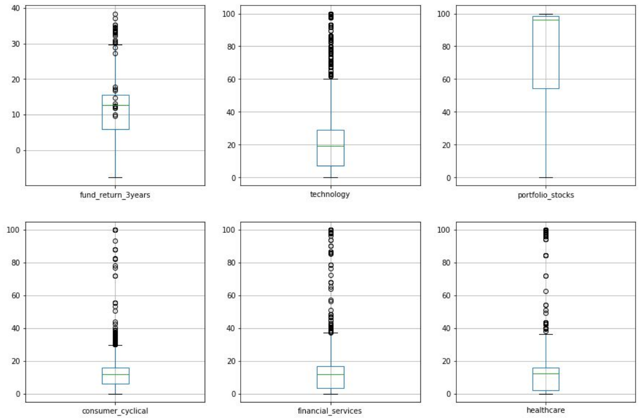
```
In [22]: df3 = df.select('fund_return_3years', 'technology', 'portfolio_stocks',
            'investment', 'consumer_cyclical', 'financial_services', 'size', 'healthcare')
         df3.limit(5)
```

Out[22]:

fund_return_3years	technology	portfolio_stocks	investment	consumer_cyclical	financial_services	size	healthcare
7.1	1.08	76.25	Blend	0.16	0.0	Large	0.0
10.09	29.99	47.13	Blend	4.78	17.68	Large	17.35
15.39	35.3	97.06	Growth	20.2	16.12	Large	16.52
9.38	21.3	75.06	Growth	14.71	16.03	Large	12.71
9.24	18.61	95.71	Blend	11.9	17.9	Large	11.23

Outlier Analysis *(used handy spark for plotting)*

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x21520c875c0>



NA Analysis

```
In [39]: from pyspark.sql.functions import isnan, when, count, col

         df3.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df3.columns]).limit(1)
```

Out[39]:

fund_return_3years	technology	portfolio_stocks	consumer_cyclical	financial_services	healthcare	inv_dummy	size_dummy	return_binary
23	22	22	22	22	22	0	0	0

Scaled Data

```
In [58]: from pyspark.ml.feature import VectorAssembler
         vecAssembler = VectorAssembler(inputCols=['technology', 'portfolio_stocks', 'consumer_cyclical',
                                                   'financial_services', 'healthcare'], outputCol="features")
         stream_df = vecAssembler.transform(df3)
```

Need to create a vector of our features

```
In [59]: stream_df.limit(5)
```

Out[59]:

technology	portfolio_stocks	consumer_cyclical	financial_services	healthcare	return_binary	features
25.38	97.8	22.86	5.93	16.87	1	[25.38,97.8,22.86...
31.89	98.48	23.76	9.48	14.95	1	[31.89,98.48,23.7...
23.5	99.99	13.44	37.53	8.51	0	[23.5,99.99,13.44...
46.82	79.05	8.16	4.93	0.0	1	[46.82,79.05,8.16...
17.58	78.89	12.65	16.61	11.73	0	[17.58,78.89,12.6...

Now we need to create a column of our scaled features

```
In [60]: from pyspark.ml.feature import MinMaxScaler
         scaler = MinMaxScaler(inputCol="features", outputCol="scaledFeatures")
         scaledData = scaler.fit((stream_df)).transform(stream_df)
         scaledData.limit(5)
```

Out[60]:

technology	portfolio_stocks	consumer_cyclical	financial_services	healthcare	return_binary	features	scaledFeatures
25.38	97.8	22.86	5.93	16.87	1	[25.38,97.8,22.86...	[0.28316411915653...
31.89	98.48	23.76	9.48	14.95	1	[31.89,98.48,23.7...	[0.35579605042954...
23.5	99.99	13.44	37.53	8.51	0	[23.5,99.99,13.44...	[0.26218899921901...
46.82	79.05	8.16	4.93	0.0	1	[46.82,79.05,8.16...	[0.52236974227379...
17.58	78.89	12.65	16.61	11.73	0	[17.58,78.89,12.6...	[0.19613968537320...

Spark SVM Model

- Initial tested parameters in spark produced *higher* metric scores (87/86%) than scikit (~85%)
- C in spark is a scaled inverse relation to scikit
- Spark does not support non-linear kernels, thus could not fine tune

SVM

Important Note

sklearn uses C to scale the loss (error) term. I.e., use a small regParam or big C to reduce regularization strength; use a big regParam or small C to increase regularization. These are (scaled) inverses of each other. The easiest way to get the best results for each implementation is to tune each separately.

In [68]:
```
from pyspark.ml.classification import LinearSVC
lsvc = LinearSVC(maxIter=10, regParam=0.1)
```

In [69]:
```
lsvcModel = lsvc.fit(train)
```

In [75]:
```
lsvcModel
```

Out[75]:
```
LinearSVCModel: uid=LinearSVC_26e03872c072, numClasses=2, numFeatures=5
```

In [70]:
```
# Print the coefficients and intercept for linearsSVC
print("Coefficients: " + str(lsvcModel.coefficients))
print("Intercept: " + str(lsvcModel.intercept))
```

```
Coefficients: [5.832556003569273,1.3477857897750258,3.373891274773128,-0.9259558304356196,3.8252590141205354]
Intercept: -3.446979066518384
```

In [73]:
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evals = MulticlassClassificationEvaluator(metricName="accuracy")
accuracy = evals.evaluate(predictions)
evals = MulticlassClassificationEvaluator(metricName="weightedPrecision")
precision = evals.evaluate(predictions)
evals = MulticlassClassificationEvaluator(metricName="weightedRecall")
recall = evals.evaluate(predictions)
print("SVM accuracy:", accuracy)
print("SVM precision:", precision)
print("SVM Recall:", recall)
print("Our ROC Score is:", evaluator.evaluate(predictions))
```

```
SVM accuracy: 0.8754098360655738
SVM precision: 0.8762023292360455
SVM Recall: 0.8754098360655738
```

In [122]:
```
print("Metrics for SVM with regParma = 0.5")
print("Our ROC Score is:", evaluator.evaluate(predictions))
evals = MulticlassClassificationEvaluator(metricName="accuracy")
accuracy = evals.evaluate(predictions)
evals = MulticlassClassificationEvaluator(metricName="weightedPrecision")
precision = evals.evaluate(predictions)
evals = MulticlassClassificationEvaluator(metricName="weightedRecall")
recall = evals.evaluate(predictions)
print("SVM accuracy:", accuracy)
print("SVM precision:", precision)
print("SVM Recall:", recall)
```

```
Metrics for SVM with regParma = 0.5
Our ROC Score is: 0.9213899252282629
SVM accuracy: 0.8688524590163934
SVM precision: 0.8691018380158062
SVM Recall: 0.8688524590163935
```

Spark Two Feature SVM & Model Comparison

- Used consumer cyclical and technology for top two feature importances
- C = 0.8-1 produced highest metrics, but performed worse than 5 features model

Train and Test Data

```
In [133]: train, test = scaledData.randomSplit([0.7, 0.3], seed = 42)
```

SVM

```
In [ ]: model = LinearSVC(maxIter=10, regParam=0.9 )
        modelfit = model.fit(train)
        predictions = modelfit.transform(test)
```

```
In [149]: print("Metrics for SVM with regParma = 0.9")
          print("Our ROC Score is:", evaluator.evaluate(predictions))
          evals = MulticlassClassificationEvaluator(metricName="accuracy")
          accuracy = evals.evaluate(predictions)
          evals = MulticlassClassificationEvaluator(metricName="weightedPrecision")
          precision = evals.evaluate(predictions)
          evals = MulticlassClassificationEvaluator(metricName="weightedRecall")
          recall = evals.evaluate(predictions)
          print("SVM accuracy:", accuracy)
          print("SVM precision:", precision)
          print("SVM Recall:", recall)
```

```
Metrics for SVM with regParma = 0.9
Our ROC Score is: 0.9023380055215683
SVM accuracy: 0.8508196721311475
SVM precision: 0.8509597426852045
SVM Recall: 0.8508196721311476
```

regParma "C"	**0.01**	**0.05**	**0.1**	**0.8**	**1**	**2**
ROC	90.6%	90.5%	91.0%	90.3%	90.0%	90.4%
Accuracy	84.8%	84.1%	78.6%	85.0%	85.1%	62.7%
Precision	84.5%	84.3%	80.4%	85.0%	85.1%	76.7%
Recall	84.8%	84.2%	78.6%	85.0%	85.1%	62.7%

Scikit vs Spark Model Comparison

- Scikit had just slightly overall higher scoring metrics than spark after optimization
- Spark optimization was limited by not allowing for optimization through non-linear kernels and gamma parameter

Scikit Models

	Model 1	Model 2 (c grid)	Model 3 (2 features, c = 5)	Model 4 (2 features, c = 0.1)	Model 5 (c grid, 2 features)
Accuracy	85.9%	88.1%	86.3%	85.8%	85.2%
Precision	85.9%	87.3%	84.8%	82.1%	85.5%
Recall	85.4%	88.7%	88.1%	91.1%	90.4%
AUC Score	85.6%	88.1%	86.4%	85.8%	83.9%

Pyspark 5 Feature Models

regParma "C"	0.1	0.5
ROC	92.3%	92.1%
Accuracy	86.7%	87.0%
Precision	87.2%	87.3%
Recall	86.7%	87.0%

Pyspark 2 Feature Models

regParma "C"	0.01	0.05	0.1	0.8	1	2
ROC	90.6%	90.5%	91.0%	90.3%	90.0%	90.4%
Accuracy	84.8%	84.1%	78.6%	85.0%	85.1%	62.7%
Precision	84.5%	84.3%	80.4%	85.0%	85.1%	76.7%
Recall	84.8%	84.2%	78.6%	85.0%	85.1%	62.7%

Other Model Comparison (SGD Classifier)

- Stochastic Gradient Descent classifier had a high precision, but much lower Recall than SVM

Cross-Validation with SGDClassifier

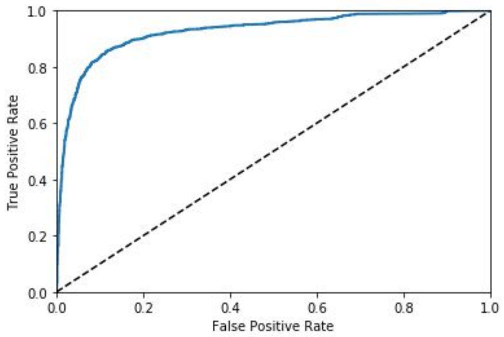
```
In [225]: from sklearn.model_selection import cross_val_score
          from sklearn.linear_model import SGDClassifier
          from sklearn.model_selection import cross_val_predict
          sgd_clf = SGDClassifier(random_state=42)
          sgd_clf.fit(X_train, Y_train)

Out[225]: SGDClassifier(alpha=0.0001, average=False, class_weight=None,
                        early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
                        l1_ratio=0.15, learning_rate='optimal', loss='hinge',
                        max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
                        power_t=0.5, random_state=42, shuffle=True, tol=0.001,
                        validation_fraction=0.1, verbose=0, warm_start=False)

In [226]: cross_val_score(sgd_clf, X_train, Y_train, cv=3, scoring="accuracy")
```

Confusion Matrix

```
In [231]: from sklearn.metrics import confusion_matrix
          from sklearn.model_selection import cross_val_predict, cross_val_score
          from sklearn import preprocessing
          import sklearn.metrics as met

In [232]: y_train_pred = cross_val_predict(sgd_clf, X_train, Y_train, cv=3)

In [233]: confusion_matrix(Y_train, y_train_pred)

Out[233]: array([[1895,  163],
                 [ 388, 1654]], dtype=int64)

In [234]: y_pred = sgd_clf.predict(X_test)
```

ROC Curve

```
In [229]: def plot_roc_curve(fpr, tpr, label=None):
              plt.plot(fpr, tpr, linewidth=2, label=label)
              plt.plot([0, 1], [0, 1], 'k--')
              plt.axis([0, 1, 0, 1])
              plt.xlabel('False Positive Rate')
              plt.ylabel('True Positive Rate')
          plot_roc_curve(fpr, tpr)
          plt.show()
```




```
SGDClassifier Metrics

Accuracy score is: 0.8498293515358362
Precision score is: 0.9006622516556292
Recall score is: 0.7825086306098964
AUC score is: 0.8490720880833509
```

Other Model Comparison (Random Forest)

- Random Forest performed extremely well, much higher than SVM
- It is also impacted by consumer cyclical but much more by technology than SVM

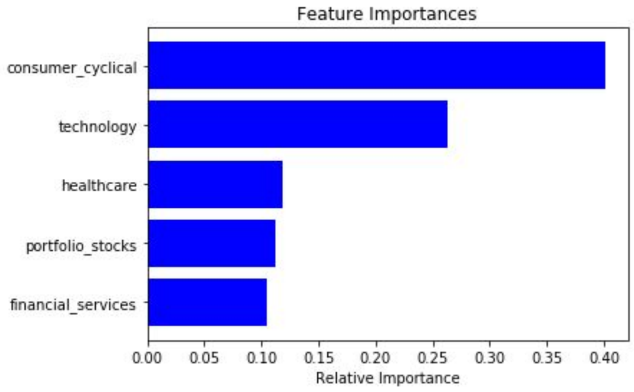
Random Forest

See what features play significant importance

```
In [236]: from sklearn.ensemble import RandomForestClassifier as  RFC

In [237]: forest = RFC(n_jobs=5, n_estimators=20, random_state =0)

In [238]: forest.fit(X_train,Y_train)

Out[238]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=5,
                       oob_score=False, random_state=0, verbose=0,
                       warm_start=False)

In [239]: y_pred = forest.predict(X_test)
```

```
Random Forest Ensemble Metrics

Accuracy score is: 0.9254835039817975
Precision score is: 0.91
Recall score is: 0.9424626006904487
AUC score is: 0.9256744949458993
```

```
In [243]: plt.figure(1)
          plt.title('Feature Importances')
          plt.barh(range(len(indices)), importances[indices], color='b', align='center')
          plt.yticks(range(len(indices)), features[indices])
          plt.xlabel('Relative Importance')

Out[243]: Text(0.5, 0, 'Relative Importance')
```



Best Tree of Random Forest

- Can use *RandomizedSearchCV* in scikit to find best parameters, or best tree

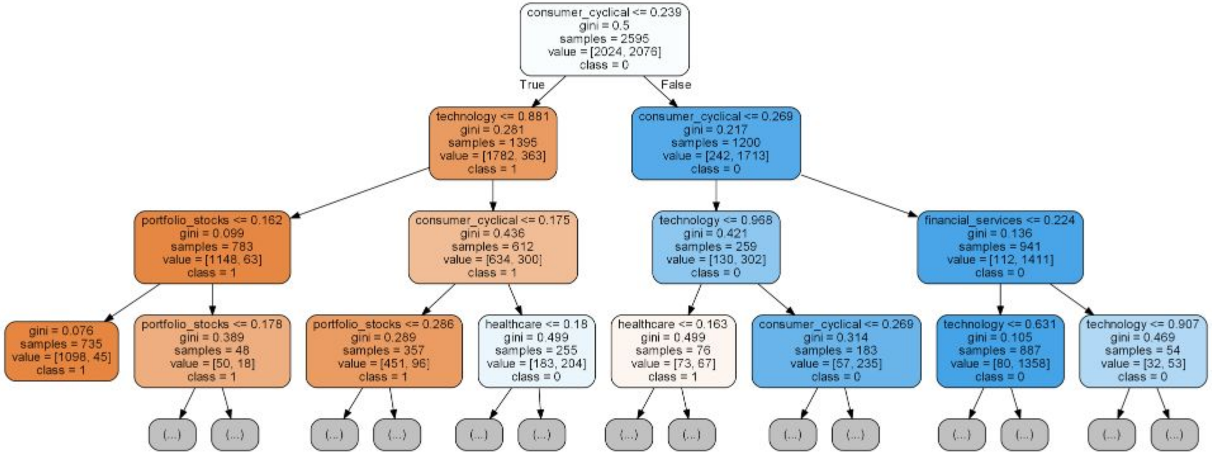
In [246]:

```
best_model = rs.best_estimator_
```

In [247]:

```
from sklearn.tree import export_graphviz
from subprocess import call
from IPython.display import Image
estimator = best_model.estimators_[1]
import os
os.environ["PATH"] += os.pathsep + r'C:\Users\fcont\Anaconda3\Library\bin\graphviz'
# Export a tree from the forest
export_graphviz(estimator, 'tree_from_optimized_forest.dot', rounded = True,
                feature_names=X_train.columns, max_depth = 3,
                class_names = ['1', '0'], filled = True)
call(['dot', '-Tpng', 'tree_from_optimized_forest.dot', '-o', 'tree_from_optimized_forest.png', '-Gdpi=200'])
Image('tree_from_optimized_forest.png')
```

Out[247]:



Conclusion

- Optimized 5 feature model for SVM produced highest metrics and just slightly outperformed Spark
- Spark does not support non-linear kernels, which could be key difference to achieving higher results

Summary

- Funds that outperform the market from 2016-2018 had characteristics of ***higher*** weight in consumer cyclical, technology, stock allocation, healthcare, and lower weighting in financial services