CMPT 423/820

Assignment 4 Question 1

- Model Solution and Grading Scheme
- 12 marks

Prologue: Importing Modules

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from sklearn.naive_bayes import GaussianNB
from sklearn import model_selection

import warnings
warnings.filterwarnings("ignore", category=UserWarning)
```

Prologue: Reading the Data

One of the data files had a different format. This was not a deliberate choice.

In [29]:
```
# a list of names for the columns in the data

filenames = ['iris', 'iris01', 'iris05', 'iris10', 'iris20']
cnames=['SepalLengthCm', 'SepalWidthCm',
        'PetalLengthCm', 'PetalWidthCm', 'Species']
# set up a dictionary to hold all the data
# they are all pretty small

dataframes = {}

# the first file format is different.  Yuck.
dataframes['iris'] = pd.read_csv('A4Q1/iris.csv', header=None,
                         names=cnames, index_col=False)

for fn in filenames[1:]:
    dataframes[fn] = pd.read_csv('A4Q1/'+fn+'.csv',
                             index_col=0)
```

Dealing with missing data

The four methods to fill in missing data are implemented as functions. For the most part, doing this part of the question was a matter of finding the right functions and methods in the Pandas API.

In [28]:
```
def remove_missing(adf):
    """ Remove any line from the dataframe that has a blank entry NA
        Returns a new dataframe
    """
    return adf.dropna()


def fill_randomly(adf):
    """ Any blank entry NA in a dataframe is filled randomly
        Returns a new dataframe
    """
    M = len(adf.index)
    N = len(adf.columns)
    ran = pd.DataFrame(np.random.rand(M,N), columns=adf.columns, index
=adf.index)

    # make a copy, because update modifies the give dataframe
    new_df = adf.copy()
    new_df.update(ran*adf.mean(), overwrite=False)
    return new_df


def fill_column_mean(adf):
    """ Any blank entry NA in a dataframe is filled with the column me
an
        Returns a new dataframe
    """
    new_df = adf.fillna(adf.mean())
    return new_df


def fill_column_class_mean(adf):
    """ Any blank entry NA in a dataframe is filled with the column me
an for that class
        Returns a new dataframe
    """

    setosa = adf[adf['Species'] == 'setosa']
    setosa = setosa.fillna(setosa.mean())

    virginica = adf[adf['Species'] == 'virginica']
    virginica = virginica.fillna(virginica.mean())

    versicolor = adf[adf['Species'] == 'versicolor']
    versicolor = versicolor.fillna(versicolor.mean())

    new_df = pd.concat([setosa,versicolor,virginica], axis=0)
    return new_df
```

Running the techniques, and checking accuracy

With these functions, we simply run all of the data files through them.

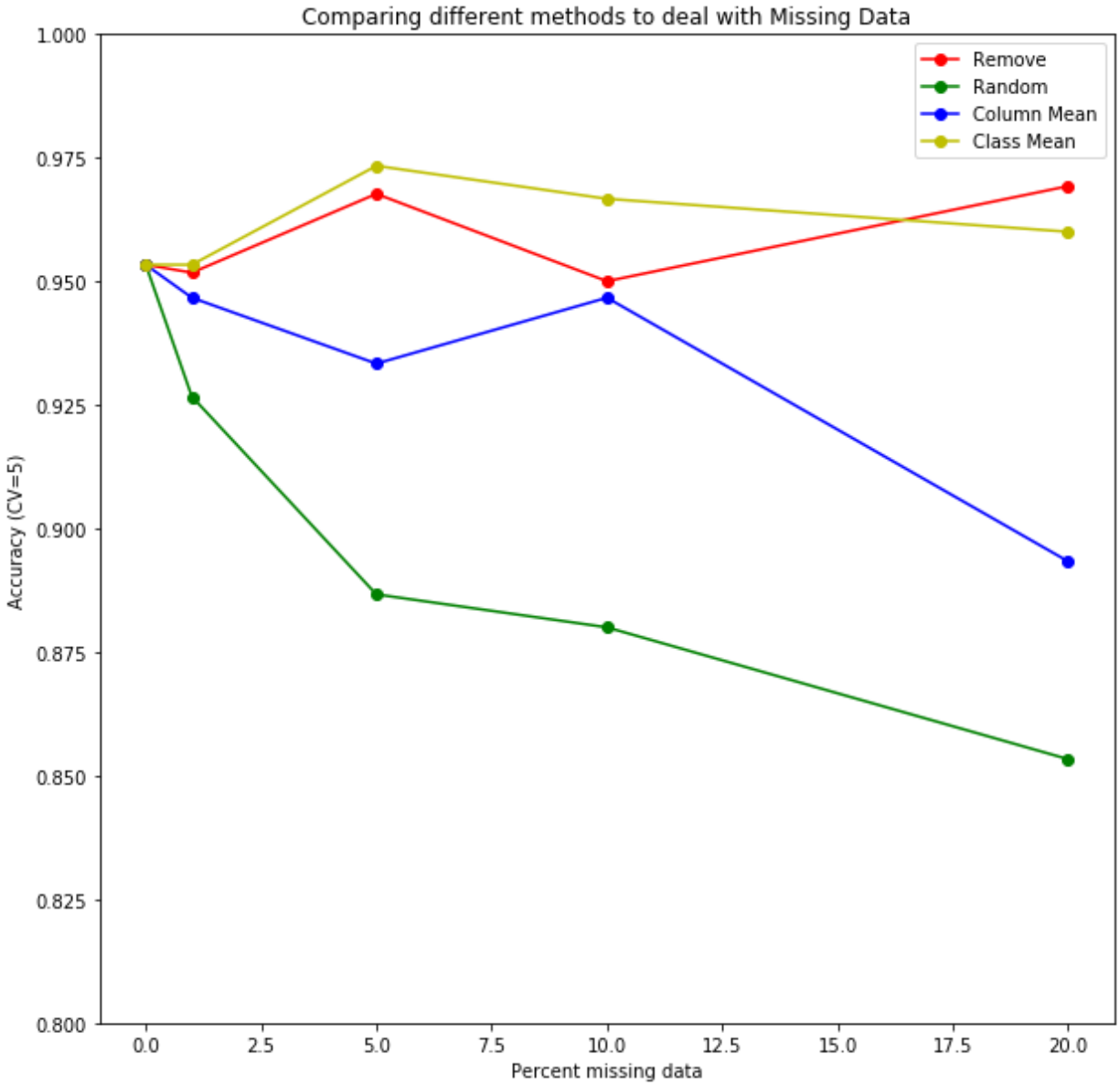
In [42]:
```
def fitNB_and_report(adf):
    """ Normal fitting of Naive Bayes with 5-fold cross-validation.
        Returns the mean and standard deviation of accuracy.
    """
    # Separate the features from the class
    array = adf.values
    X = array[:,0:4]
    Y = array[:,4]

    # cross validation to evaluate the performance using accuracy
    kfold = 5
    cv_results = model_selection.cross_val_score(GaussianNB(), X, Y, c
v=kfold, scoring='accuracy')
    return cv_results.mean(), cv_results.std()

removed           =  [fitNB_and_report( remove_missing(dataframes[fn])
)         for fn in dataframes]
mean_filled       =  [fitNB_and_report( fill_column_mean(dataframes[fn
]))       for fn in dataframes]
class_mean_filled =  [fitNB_and_report( fill_column_class_mean(datafra
mes[fn])) for fn in dataframes]
randomly          =  [fitNB_and_report( fill_randomly(dataframes[fn]))
for fn in dataframes]
```

In [45]:
```
percent_missing = [0, 1, 5, 10, 20]

plt.figure(figsize=(10,10))
plt.plot(percent_missing, [m for m,s in removed],           'ro-')
plt.plot(percent_missing, [m for m,s in randomly],          'go-')
plt.plot(percent_missing, [m for m,s in mean_filled],       'bo-' )
plt.plot(percent_missing, [m for m,s in class_mean_filled], 'yo-')
plt.ylim(0.8, 1.0)
plt.title("Comparing different methods to deal with Missing Data")
plt.ylabel('Accuracy (CV=5)')
plt.xlabel('Percent missing data')
plt.legend(['Remove', 'Random', 'Column Mean', 'Class Mean'])
plt.show()
```

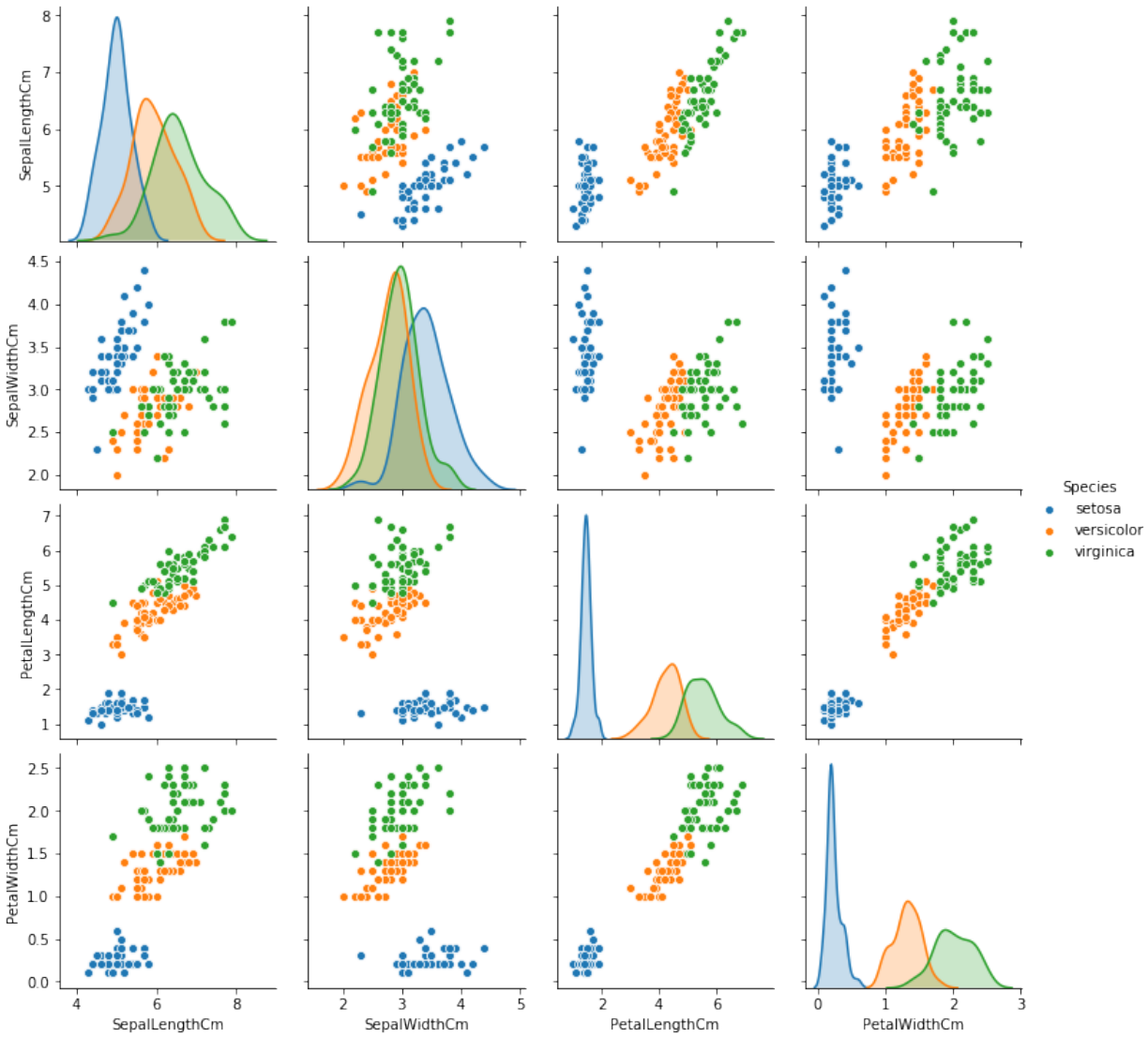


Commentary

We can understand this result by considering what the different techniques do to data points. If you think back to A2, you might remember the Pair-Plot figure showing each pair of features plotted in 2D. The classes appeared as clouds of points. Sometimes the clouds were separated, and sometimes they were over-lapping, to various degrees.

Here's a reminder of those plots. Notice the clouds of data. Imagine slidine any particular point one a line parallel to one of the axes. We'll need that idea below.

In [44]:
```
sns.pairplot(dataframes['iris'], hue="Species", diag_kind='kde')
plt.show()
```



Let's consider each technique individually.

1. **Remove**: This technique simply reduced the number of points. Because the process I used to put holes into the data was essentially random, the missing data was roughly equally split between the three classes. There were fewer data points in each cloud of points. If you take enough points away, you begin to take away spome of the points that resulted in over-lap. Eventually, classes will be easier to classify. But this result is over-fitting: the nice neat classes that result from just a few data points, are sure to make mistakes on the points we know exist in the regions of over-lap. This could be shown using a validation set.
2. **Replace with random.** When we fill a hole with a random data point, we are essentially randomly sliding the point somewhere along the missing column. It's like taking the original data, and sliding the point somewhere parallel to one of the axes in the Pair-Plot. The resulting plots will not form nice neat clouds, and a classifier will have a hard time classifying.
3. **Replace with column mean.** When we fill a hole with the column mean, we are essentially sliding the point to the middle of one of the axes in the Pair-Plot. IT's better than random, but it can result in a central cluster of points that are not the same class.
4. **Replace with column mean for the given class.** When we fill a hole with the class mean, we are essentially sliding the point to the middle of one of the clusters in the Pair-Plot. This is a lot better than the middle of the axis, and it's a lot better than a random place on the axis.

I think it's not surprising that "Random" and "Column Mean" were not as good as "Class Mean." However, it might be surprising that throwing away data results in good classifier accuracy. Being able to see this as a form of over-fitting is key to understanding why we want to avoid it.

Grading -- 12 marks

- 8 marks. Your plot showed the data for the 4 techniques and the 5 data files.
  - It doesn't have to be a single plot. It could be several.
  - The plots are consistent with the solution here.
- 4 marks. You discussed and interpreted the results.
  - You noted that random was bad, and class mean was good.
  - You explained why removing data worked well.
  - There was no need to reproduce the pair plots in the explanation.

CMPT 423/820

Assignment 4 Question 2

- Marking Scheme and Grading Guidelines
- 24 marks

Importing common libraries ¶

In [1]:
```
%matplotlib inline
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Importing other libraries

We'll import the SVM library, and the library that lets us work with artificial and built-in datasets.

In [3]:
```
from sklearn import svm
from sklearn import datasets
```

A function to visualize how a classifier works on given data

The function below is basically MatPlotLib magic to make nice visualizations of the SVM classifier for simple 2D data. It fits the given classifier, and retrieves the support vectors from it afterward. As a result, it's specialized to 2D SVM models.

The separating hyperplane is a solid line, and the two margins are visualized as dashed lines. The support vectors, i.e., the data points that contribute towards the location of the separating hyperplane, are visualized as circled points. Data that are not circled do not affect the location of the hyperplane.

You can call this function without having to understand everything it's doing.

In [4]:
```
def plot_hyperplane(svm_clf, X, y):
    """ Plot the separating hyperplane determined by svm_clf on data X
with classes y.
        The classifier svm_clf is assumed not to have been fitted to t
he data already.
        Data is assumed to be 2D, with exactly 2 class labels.
    """
    # first do the fitting
    svm_clf.fit(X, y)

    # plot the data values X
    plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

    # plot the decision function
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    xx = np.linspace(xlim[0], xlim[1], 30)
    yy = np.linspace(ylim[0], ylim[1], 30)
    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T
    Z = svm_clf.decision_function(xy).reshape(XX.shape)

    # plot decision boundary and margins
    ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
               linestyles=['-.', '-', '--'])
    # plot support vectors
    ax.scatter(svm_clf.support_vectors_[:, 0], svm_clf.support_vectors
_[:, 1], s=100,
               linewidth=1, facecolors='none', edgecolors='k')
    plt.show()
```

Generate some synthetic data

We can generate some synthetic data using some built in tools provided by scikit in the library `sklearn.datasets`.

- `make_blobs()` creates separable clusters in 2D.
- `make_circles()` creates a blob with another set of samples in a circle around it.
- `make_moons()` creates semi-circles that are not linearly separable

For the `plot_hyperplane()` function above, we're limited to two classes for the visualization. To generate new data change the `random_state` value.

These datasets are 2D, so we can visualize them.

In [5]:
```
blob_X, blob_y = datasets.make_blobs(n_samples=40, centers=2, random_s
tate=34)
circ_X, circ_y = datasets.make_circles(n_samples=40, random_state=84)
moon_X, moon_y = datasets.make_moons(n_samples=40, random_state=61)
```

Regularization in SVM models

In class, we approached regularization by formulating a Loss function that balanced error against the complexity of the model.

$$Loss(x, m) = Err(x, m) + \lambda Complexity(m)$$

When λ gets bigger, the Complexity of the model contributes more to the Loss, and simpler models are emphasized. When λ gets big enough, the compromise between cost and complexity may incur increased error, just to keep the complexity cost down.

It turns out that some machine learning models, including SVM, traditionally use a different formulation:

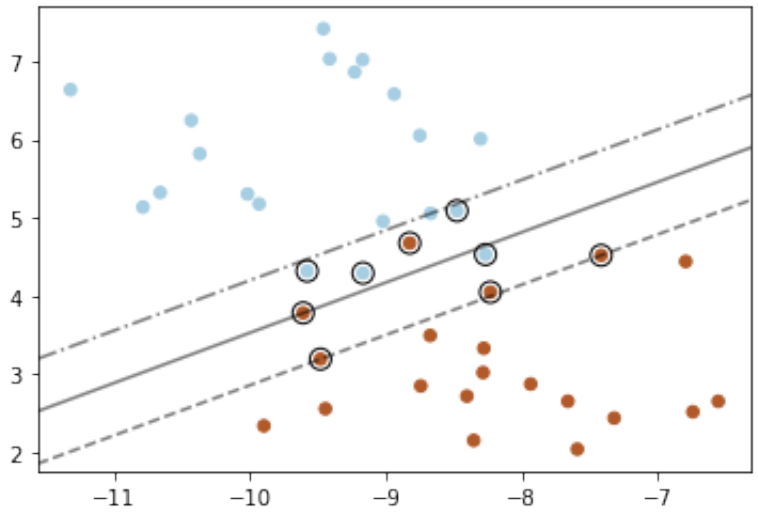
$$Loss(x, m) = C \times Err(x, m) + Complexity(m)$$

In this formulation, as C gets smaller, the Complexity of the model contributes more to the Loss, and simpler models are emphasized.

Either way, it's the relative balance between error and complexity that matters. In the scikit libraries, we can indicate how to balance error and complexity using the keyword parameter `C=1.0` To emphasize Error, make `C>>>1`. To emphasize complexity, make `C<<1`.

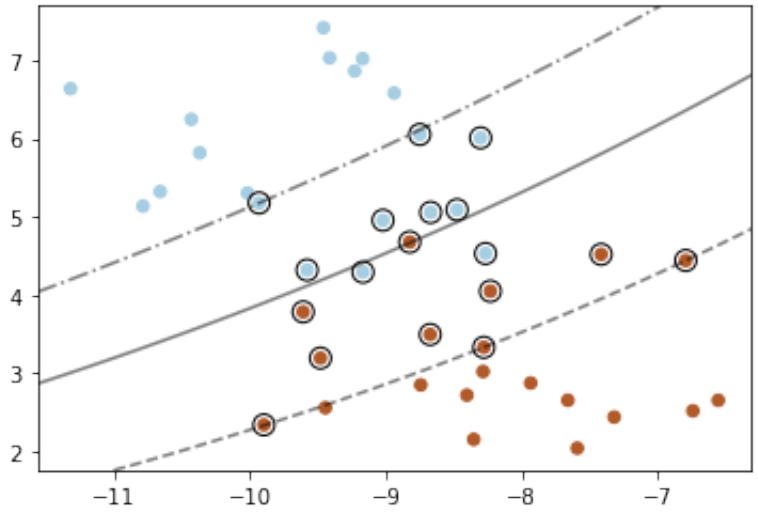
Apply a simple linear SVM to some data

In [6]:
```
plot_hyperplane(svm.SVC(kernel='linear', C=1), blob_X, blob_y)
```



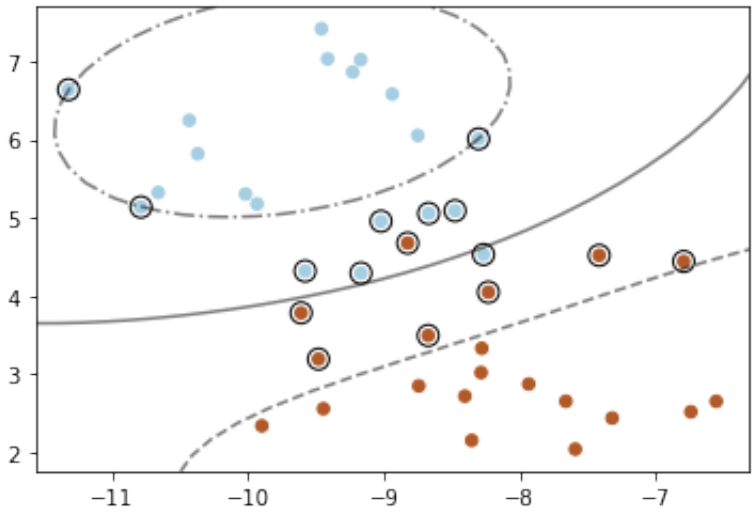
Apply SVM with a Polynomial Kernel to some data

In [7]:
```
plot_hyperplane(svm.SVC(kernel='poly', degree=3, C=1), blob_X, blob_y)
```



Apply SVM with a Radial Basis Function Kernel to some data

In [8]:
```
plot_hyperplane(svm.SVC(kernel='rbf', gamma=0.1, C=1), blob_X, blob_y)
```



Part 1 --- Kernels!

Describe the term *kernel* in the context of Support Vector Machines. Explain the polynomial kernel, and the Radial Basis function kernel.

Part 1 Answer

In general, a *kernel* is a function that takes two vectors as arguments, and produces a scalar value. There are mathematical restrictions on kernels: including that they have to be symmetric in their arguments; the textbook (Bisho Ch 6.2 talks about kernels nand their properties).

A kernel function can be applied to data samples, but is not limited to this usage. When applied to data samples, we can interpret the kernel as performing a dot-product of the two data samples in a new feature space. The definition is as follows:

$$k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^{\mathrm{T}} \phi(\mathbf{y})$$

In this definition, $x^T y$ represents the dot-product (also called "inner product"). The function $\phi()$ is any function that maps a vector from one space to another. The function ϕ can map to a higher or lower dimensional space, or can warp the coordinates, or almost anything at all, as long as it produces a new vector.

The **linear kernel** is defined by

$$k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^{\mathrm{T}} \mathbf{y}$$

that is, just the normal dot-product, with no prior transformation to another space (ϕ is the identity function $\phi(x) = x$)

The **polynomial kernel** is defined by

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^{\mathrm{T}} \mathbf{y})^d$$

It starts with the same dot-product as the linear kernel, but raises the result to a constant power, d. In some formulations, a constant c is added to the dot-product prior to exponentiation. The polynomial kernel has the effect of stretching large scalars (i.e., all x where $|x| > 1$), and shrinking small ones (i.e, all x where $|x| < 1$).

Since polynomial kernel uses the dot-product, we can interpret this further. The dot product $x^T y$ will be higher when x and y are more or less along the same direction, and closer to zero when they are in orthogonal directions (i.e., at right angles).

The **radial basis kernel** is defined by

$$k(\mathbf{x}, \mathbf{y}) = \frac{\exp(-||\mathbf{x} - \mathbf{y}||^2)}{\gamma}$$

where γ is a positive constant. Sometimes also called the Gaussian kernel, in which case γ is usually replaced by somethien that looks more like what we'd see in the Gaussian probability density function: $2\sigma^2$.

This looks quite different from the linear and polynomial kernels. Notice that the bigger the difference between x and y, the smaller the result of the kernel function. Unlike the normal dot-product, which is concerned with direction, the radial basis kernel is concerned with location.

Part 1 Grading -- 6 marks

Grading should be generous here. Deduct marks only when effort was lacking, or if descriptions are quite wrong.

1. 6 Marks **Kernels!**

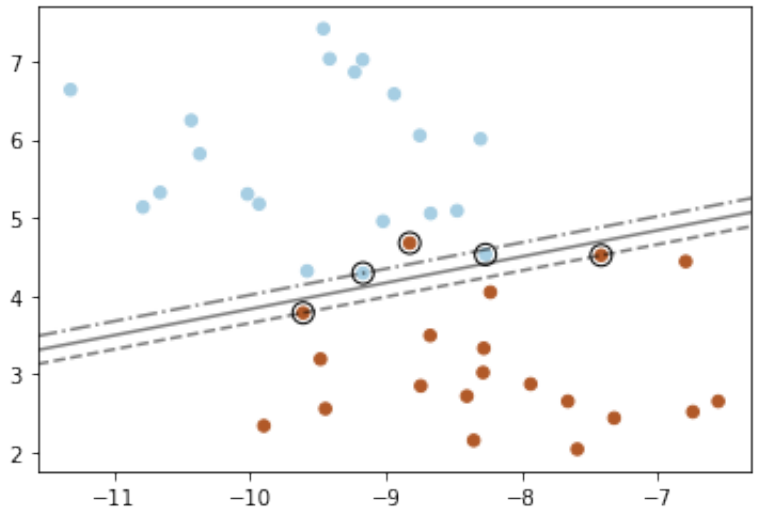
- (2 marks) You description of the term *kernel* demonstrated understanding.
  - To get full marks, the description indicates that it maps pairs of vectors (or data samples) to a single scalar value. Presentation of the equation (as above) satisfies this requirement, but is not required. The idea of mapping to a feature space is also good, but not required. A description of the linear kernel is not required.
- (2 marks) Your desciption of the polynomial kernel demonstrated understanding.
  - To get full marks, the description should indicate that the result of the normal dot-product is raised to a constant power. Presentation of the equation (as above) satisfies this requirement, but is not required. Any sort of description that attempts to interpret the kernel geometrically or mathematically is valid, unless it's quite wrong. Some examples of acceptable ideas:
    - A straight line in the polynomial feature space is likely to look curved in normal space;
    - A straight line in normal space will look like a degree d polynomial in the feature space.
- (2 marks) Your desciption of the Radial Basis Function kernel demonstrated understanding.
  - To get full marks, the description should indicate that the RBF kernel uses the $\exp$ function, and the difference between two points or vectors. Presentation of the equation (as above) satisfies this requirement, but is not required. Any sort of description that attempts to interpret the kernel geometrically or mathematically is valid, unless it's quite wrong. Some examples of acceptable ideas:
    - RBF expresses similarity by distance
    - RBF looks for circles
    - Points x and y that are on the same circle in normal space will be mapped to a common value.

Part 2 --- Exploring the balance between error and model complexity

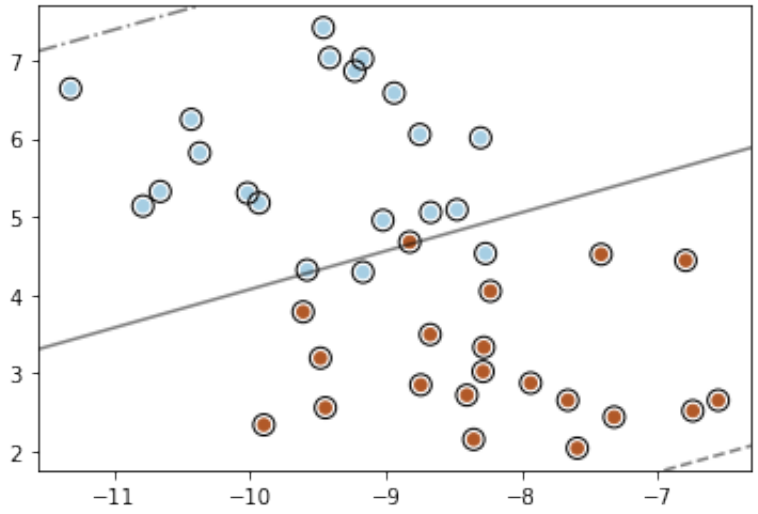
Explore what happens to the three models (`linear` , `poly` , `rbf`) when you change `C` . Use `C=1000` and `C=0.001` **on the blob dataset.** Your answer may depend on the dataset you are using (so if you change the `random_state` , the behaviour may change).

Apply a simple linear SVM to some data

In [9]:
```
plot_hyperplane(svm.SVC(kernel='linear', C=1000), blob_X, blob_y)
```



In [10]:
```
plot_hyperplane(svm.SVC(kernel='linear', C=0.005), blob_X, blob_y)
```



Discussion

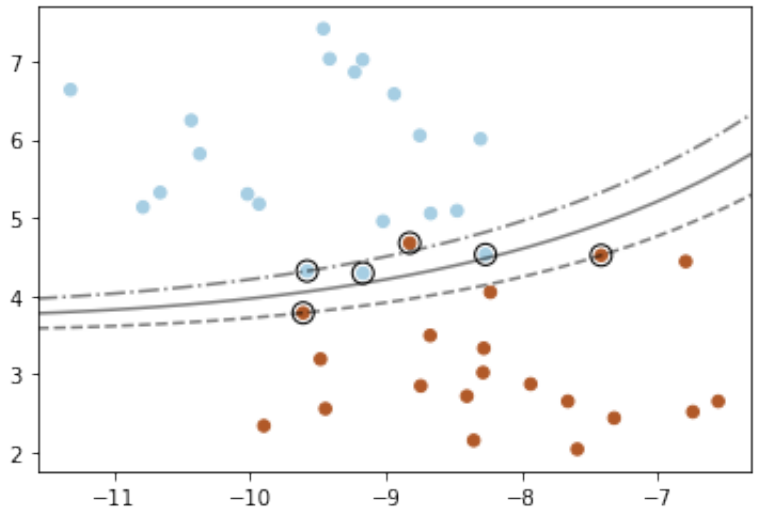
When C is quite high, the error term dominates the loss function. This results in a narrow margin, and a few support vectors (circled data points). This is all nicely visible in the plots above.

When C is quite small, the error is almost irrelevant, and the complexity dominates the loss function. In the plot above, we see lots of support vectors. And we see one of the three lines. The other two lines are not visible in the scale of the plot.

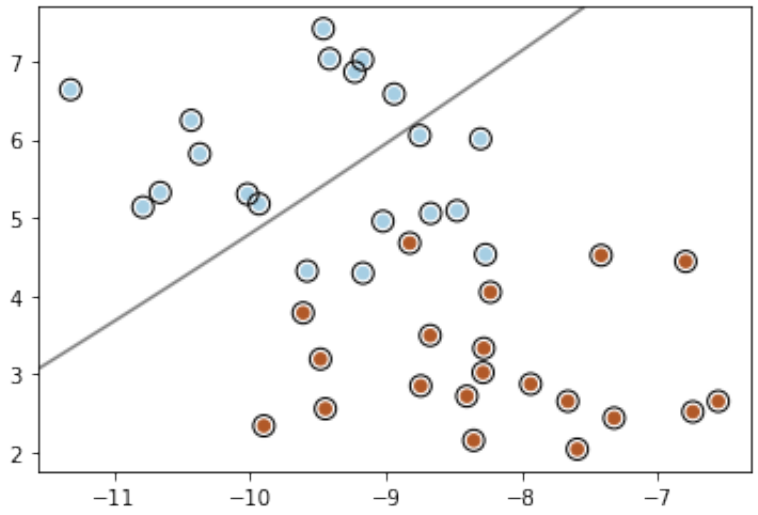
It appears that the visible line here is one of the margin lines, but I think that's a bug in the `plot_hyperplane()` function. If we decrease the value of C gradually, what we see is the widening of the margins, with the separating hyper-plane remaining more or less fixed. Only when the two margin lines are pushed outside of the scale represented above do we see the sudden change in line style where the separating hyperplane used to be.

Apply SVM with a Polynomial Kernel to some data

In [11]:
```
plot_hyperplane(svm.SVC(kernel='poly', degree=3, C=1000), blob_X, blob_y)
```



In [12]:
```
plot_hyperplane(svm.SVC(kernel='poly', degree=3, C=0.001), blob_X, blo
b_y)
```



Discussion

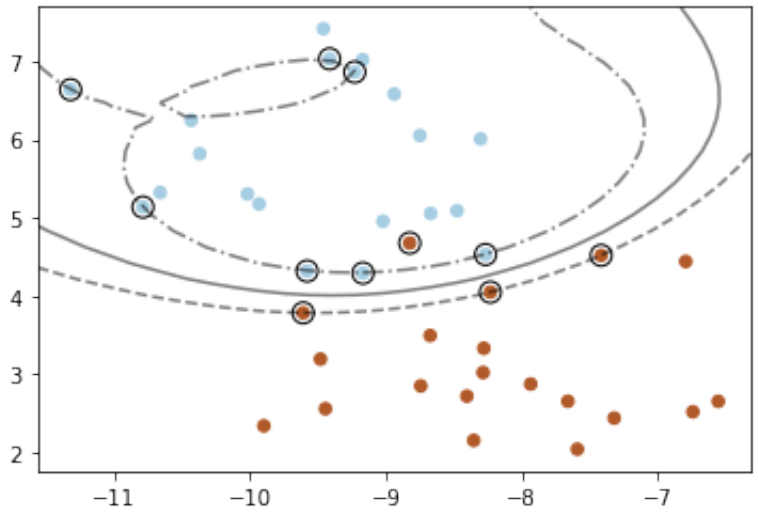
When C is quite high, the error term dominates the loss function. This results in a narrow margin, and a few support vectors (circled data points). This is all nicely visible in the plots above.

When C is quite small, the error is almost irrelevant, and the complexity dominates the loss function. In the plot above, we see lots of support vectors. And we see one of the three lines. The other two lines are not visible in the scale of the plot.

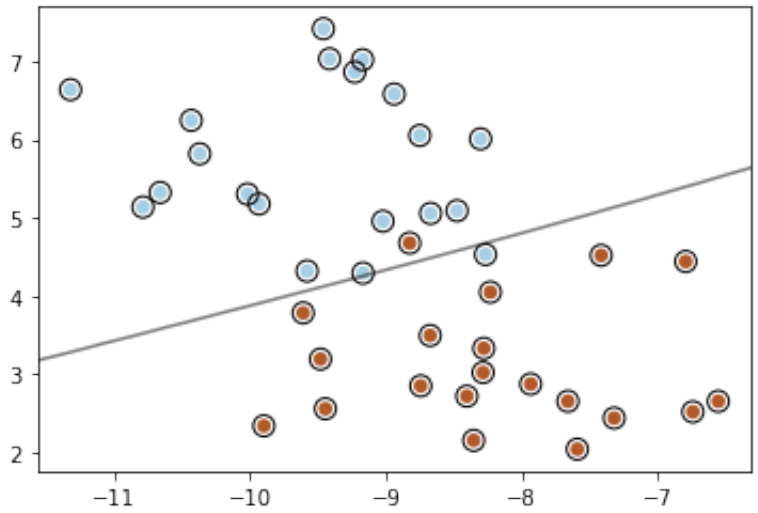
One of the observations that could be made is that the lines appear to bend more when C is large, and less when C is small. The separating hyperplane seems to emphasize more extreme support vectors when trying to minimize error, but emphazes more sensible support vectors when minimizing the complexity of the hyperplane.

Apply SVM with a Radial Basis Function Kernel to some data

In [13]:
```
plot_hyperplane(svm.SVC(kernel='rbf', gamma=0.1, C=1000), blob_X, blob
_y)
```



In [14]:
```
plot_hyperplane(svm.SVC(kernel='rbf', gamma=0.1, C=0.001), blob_X, blo
b_y)
```



Discussion

When C is quite high, the error term dominates the loss function. This results in a narrow margin, and a few support vectors (circled data points). The RBF kernel is showing its uniqueness, in the elliptical nature of the margins.

When C is quite small, the error is almost irrelevant, and the complexity dominates the loss function. In the plot above, we see lots of support vectors. And we see one of the three lines. The other two lines are not visible in the scale of the plot.

The line that we do see is the separating hyperplane, and it's as close to straight as a curve can be.

Part 2 Grading --- 6 marks

I think the bug in `plot_hyperplanes()` was not obvious (it's a consequence of matplotlib, not the script), so it's okay if student responses do not correctly identify when a single line is the actual separating hyperplane, and call it a margin line (because that's what it looks like).

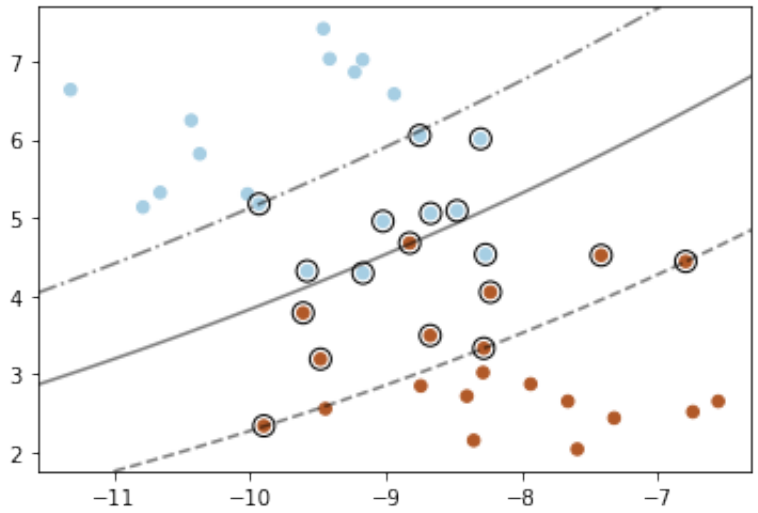
1. **Balance between error and model complexity**. 6 Marks

    - (2 marks) Your discussion on the behaviour of the `linear` kernel as `C` changes reflects an understanding of SVM.
        - The main point in all of this is that reducing error means narrowing the margins, and reducing model complexity results in broadening of the margins.
    - (2 marks) Your discussion on the behaviour of the `poly` kernel as `C` changes reflects an understanding of SVM.
        - Lines get curvier with large C, in addition to narrower margins.
    - (2 marks) Your discussion on the behaviour of the `rbf` kernel as `C` changes reflects an understanding of SVM.
        - Lines get curvier with large C, in addition to narrower margins.

Part 3 --- Polynomial Kernel Exploration

Explore the Polynomial kernel, by using the optional keyword parameter `degree` which is available in scikit, **on the blob dataset.** Use a couple of different integers for the degree (in the range 2-5, to limit computation times), and explain what happens. You may explore other `svm.SVC` parameters in the context of exploring degree, but this is optional.

In [15]:
```
plot_hyperplane(svm.SVC(kernel='poly', degree=3, C=1), blob_X, blob_y)
```



As the degree of the polynomial increases, the separating hyperplane changes shape.

- $d = 2$: a line gently curving up to the right
- $d = 3$: a curve folding up and back to the right
- $d = 4$: a curve in the form of a stretched out S
- $d = 5$: a stretched out S-curve, but the margins are very narrow.

Part 3 Grading --- 3 Marks

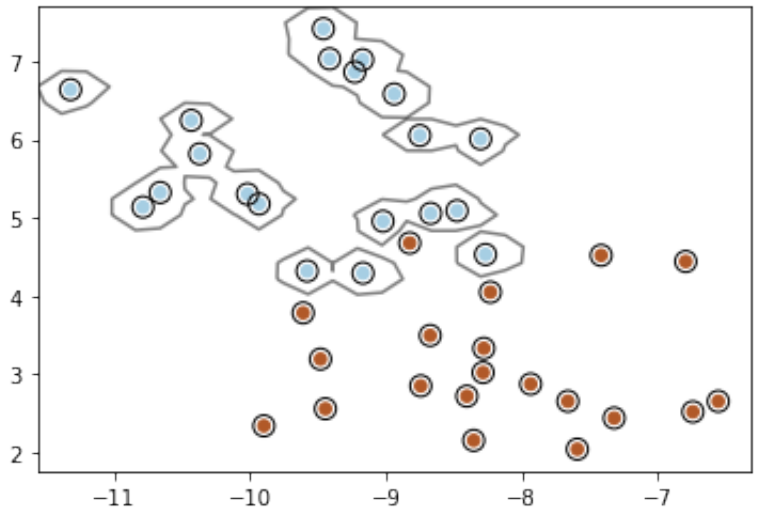
1. **Explore the Polynomial kernel** 3 marks

   - (3 marks) Your explanation of the behaviour of the polynomial kernel with different degrees reflects an understanding of SVM.
     - For full marks, some attempt to describe how the degree parameter (`degree` in the Python, d in the definition above) affects the separating hyperplane.

Part 4 --- RBF Kernel Exploration

Explore the Radial Basis Function kernel, by using the optional keyword parameter `gamma`, **on the blob dataset.** Try various extreme values for `gamma`, and explain what happens. You may explore other `svm.SVC` parameters in the context of exploring degree, but this is optional.

In [16]:
```
plot_hyperplane(svm.SVC(kernel='rbf', gamma=100, C=1), blob_X, blob_y)
```



As `gamma` changes, the separating hyperplane changes shape.

- $\gamma = 0.01$: the separating hyperplane is almost straight, the margin lines are curved slightly away.
- $\gamma = 0.1$: the separating hyperplane is curving upwards; one of the margin lines is an ellipse.
- $\gamma = 1$: the separating hyperplane is a complex curve, folding back on itself; the margin lines are are closed curves shaped to fit the example data.
- $\gamma = 10$: the separating hyperplane is a complex curve completely surrounding one of the data sets. The margin lines are expressed as a collection of tight closed curves around the data samples.
- $\gamma = 100$: It looks like only the separating hyperplane is being displayed here (same bug as mentioned earlier). The separating hyperplane is a complex curve completely surrounding different subsets of one of the class examples. The margin lines are not visible.

In general, the higher γ, the more the separating hyperplane is over-fitting to the data set.

Part 4 Grading --- 3 marks

1. **Explore the Radial Basis Function kernel** 3 marks

   - (3 marks) Your explanation of the behaviour of the Radial Basis Function kernel with different degrees reflects an understanding of SVM.
     - For full marks, some attempt must be made to describe how the `gamma` parameter affects the shape of the separating hyperplane.
     - it would not be wrong to mention over-fitting here, but it is not required for full marks.

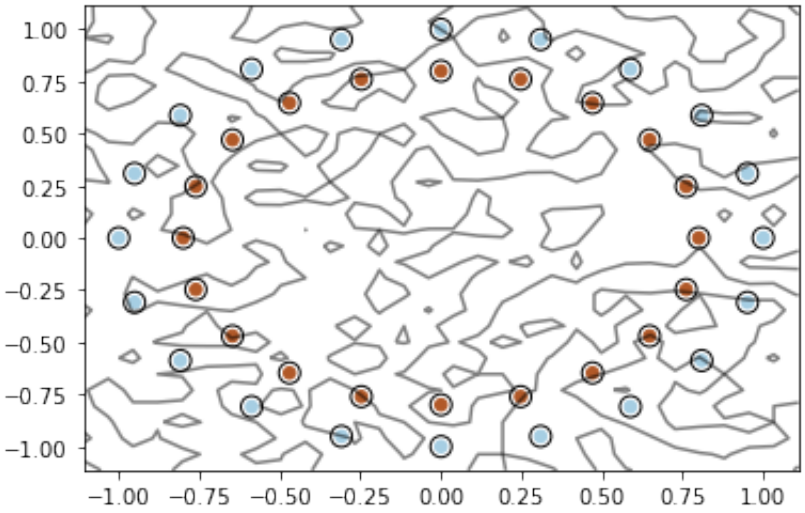
Part 5 --- Other synthetic datasets

Explore all three different kernels on the two other datasets above. Choose a parameter setting for each kernel, including a value for `C`, that seems to be lead to what you consider a good fit. Explain briefly why you decided on the `svm.SVC` parameter settings that you chose.

Part 5 Answer

Circle Data Set

In [17]:
```
plot_hyperplane(svm.SVC(kernel='linear', C=1), circ_X, circ_y)
```

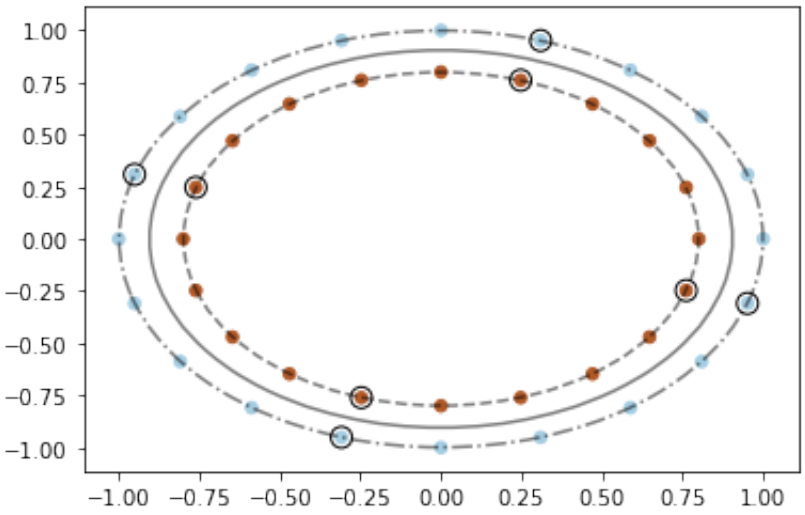


I used the default value for C, and there are no other parameters for the linear kernel SVM. The result is very odd. We see only one line, which is presumably the separating hyperplane. But we expect a straight line in this case, which definitely didn't happen. No line will do well with this data set, and SVC seems to be converging on a "line" with zero slope and zero intercept, but not quite zero due to the limited precision of computer arithmetic. That means every point in the plane should be zero in principle, but in practice, the points are only more or less zero. The line shows where zero was reached. It turns out to be noise.

I would say that the linear kernel is inappropriate for this dataset, or in other words, there is no good setting.

Apply SVM with a Polynomial Kernel to some data

In [18]:
```
plot_hyperplane(svm.SVC(kernel='poly', degree=2, C=10), circ_X, circ_y
)
```



I explored even and odd degree (d). The odd degree results were similar to the linear kernel, with a noisy set of lines and closed curves visualized as as separating hyperplane. These were rejected as inappropriate.

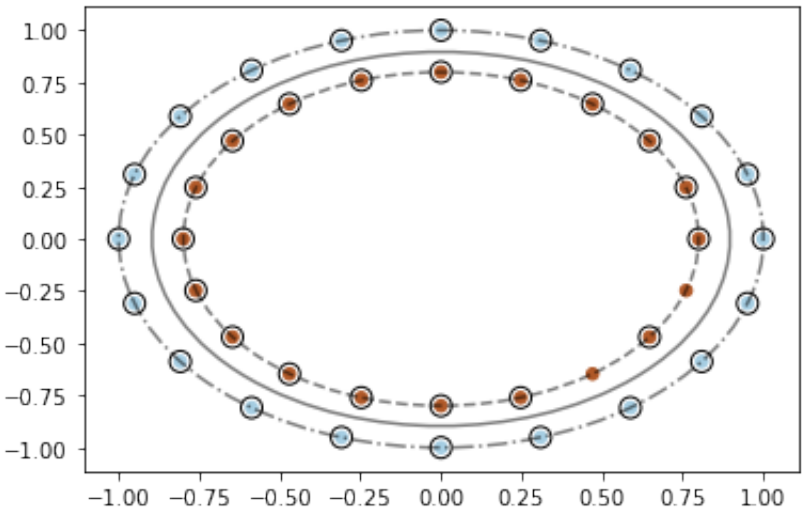
The even degree results were promising, showing a nice circular separating hyperplane between the two classes. The picture did not chnge much with increasing even d, so I am claiming $d = 2$ as an appropriate selection. These really are circles, even though they look like ellipses!

Increasing C here leads to narrower margins. When $C = 10$, the margins get pretty clode to the true separation between the two classes. Increasing $C > 10$ changes only the number of support vectors.

My final setting is $d = 2, C = 10$.

Apply SVM with a Radial Basis Function Kernel to some data

In [19]:
```
plot_hyperplane(svm.SVC(kernel='rbf', gamma=1, C=10), circ_X, circ_y)
```



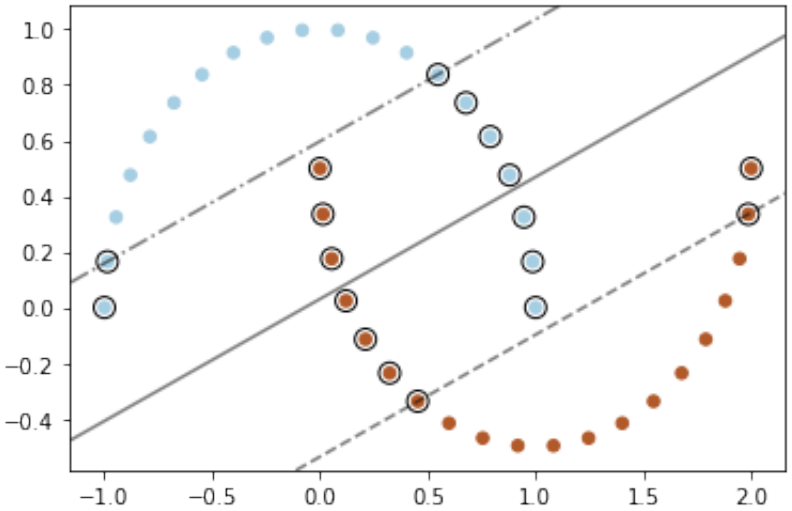
Looking at the scale of the plot, we can see that the circles have radius close to 1. If the data were more spread out, using a γ that scales the data to a unit circle is probably useful. Setting $\gamma = 1$ yields nice circles that don't improve with higher γ. If γ gets too high, we begin to see concentric circles and other weirdness.

Setting $C = 10$ brought the margins very close to the true separation for of the two classes.

My final setting is $\gamma = 1, C = 10$.

Moon Data Set

In [20]:
```
plot_hyperplane(svm.SVC(kernel='linear', C=1), moon_X, moon_y)
```

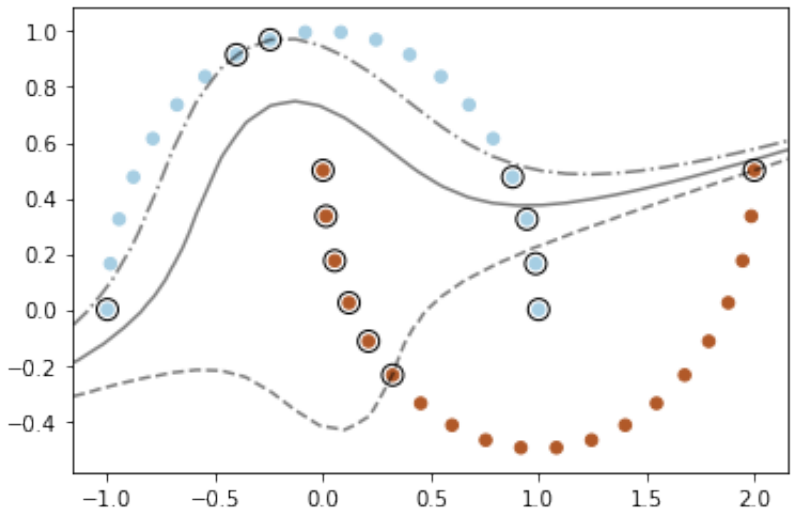


I used the default value for C, and there are no other parameters for the linear kernel SVM. The resulting separating hyperplane is about as good as any straight line could get. Increasing C didn't seem to have an effect.

If linear SVM were the only choice, we'd have to live with this kind of error, even though there is a pattern evident here.

Apply SVM with a Polynomial Kernel to some data

In [21]:
```
plot_hyperplane(svm.SVC(kernel='poly', degree=3, C=10), moon_X, moon_y
)
```



I initially kept $C = 1$ fixed, and explored even and odd degree (d).

The even degree results created symmetric hyperplanes that seemed to classify only a small handful of the data points correctly. These were rejected as inappropriate. The odd degree results equally unpromising, until I tried higher C.

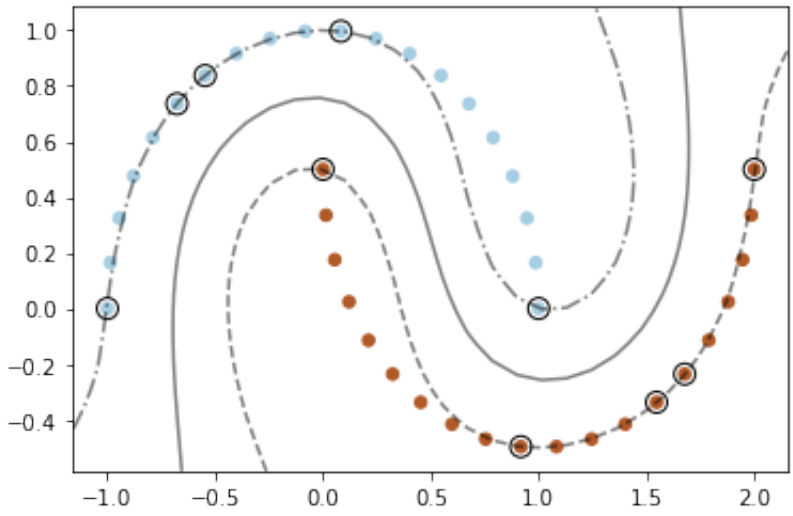
Using $C = 10$ emphasizes error over keeping model complexity low. With this setting, the $d = 3$ separating hyperplane seemed to find a curve that could keep all of the browish points (lower) classified correctly, and most of the upper points (but not all).

Increasing C beyond $C = 10$ did narrow the magins, but made the margin lines look contrived and over-fitted. Increasing d to odd degrees beyond $d = 3$ made the separating hyperplane fit more closely to the upper data set, and seemed to be avoiding the lower set too much.

My final setting is $d = 3, C = 10$.

Apply SVM with a Radial Basis Function Kernel to some data

In [22]:
```
plot_hyperplane(svm.SVC(kernel='rbf', gamma=1, C=10), moon_X, moon_y)
```



I started with the setting $\gamma = 1, C = 10$ from the circle dataset. This setting created a nice neat curve that correctly separated all the data. The margin lines pretty faithfully expressed the true separation of the data points.

Reducing to $\gamma = 0.1, C = 10$ ruined the shape of the separating hyperplane, though increasing C seems to help improve the shape of the curve again.

Increasing to $\gamma = 3, C = 10$ kept the shape of the separating hyperplane, but produced margin lines that consisted of multiple closed curves around subsets of the data points.

My final setting was $\gamma = 1, C = 10$.

Part 5 Grading --- 6 marks

1. 6 marks **Other datasets**

- (3 marks) Your explanation for your choice of parameters for all three kernels reflects an understanding of the application of SVM to the `circ` dataset created by `make_circles`.
  - full marks for each kernel if a justification was given.
  - Partial credit if just settings were given.
    - There is no good setting for linear SVC. The weird plot is evidence of something not working right, but it may not have been obvious.
    - Even degree polynomials should work well with circular data.
    - Non-extreme values for γ should be preferred.
- (3 marks) Your explanation for your choice of parameters for all three kernels reflects an understanding of the application of SVM to the `moon` dataset created by `make_moons`.
  - full marks for each kernel if a justification was given.
  - Partial credit if just settings were given.
    - There is no good setting for linear SVC. But at least the plot makes sense.
      - Odd degree polynomials work best.
      - Non-extreme values for γ should be preferred.

Marking Scheme

1. 6 Marks **Kernels!**

    - (2 marks) You description of the term *kernel* demonstrated understanding.
    - (2 marks) Your description of the polynomial kernel demonstrated understanding.
    - (2 marks) Your description of the Radial Basis function kernel demonstrated understanding.

1. 6 Marks **Balance between error and model complexity**.

    - (2 marks) Your discussion on the behaviour of the `linear` kernel as `C` changes reflects an understanding of SVM.
    - (2 marks) Your discussion on the behaviour of the `poly` kernel as `C` changes reflects an understanding of SVM.
    - (2 marks) Your discussion on the behaviour of the `rbf` kernel as `C` changes reflects an understanding of SVM.

1. 3 marks **Explore the Polynomial kernel**

    - (3 marks) Your explanation of the behaviour of the polynomial kernel with different degrees reflects an understanding of SVM.

1. 3 marks **Explore the Radial Basis Function kernel**

    - (3 marks) Your explanation of the behaviour of the Radial Basis Function kernel with different degrees reflects an understanding of SVM.

1. 6 marks **Other datasets**

    - (3 marks) Your explanation for your choice of parameters for all three kernels reflects an understanding of the application of SVM to the `circ` dataset created by `make_circles`.
    - (3 marks) Your explanation for your choice of parameters for all three kernels reflects an understanding of the application of SVM to the `moon` dataset created by `make_moons`.

CMPT 423/820

Assignment 4 Question 3

- Solutions and Grading Guide
- 12 marks

First some libraries...

In [1]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Step 1

The `a4q3.csv` file is a copy of one of the data files we used earlier in the term. It has 4 columns: `index, x, y, L`, where `L` represents a label. The `x, y` are continuous quantities, good for plotting in 2D. We will only use `L` to colourize our figures.

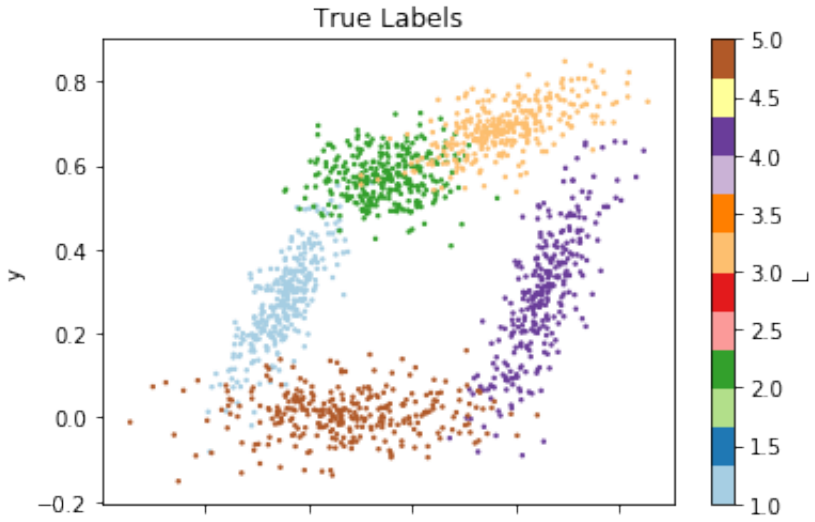
In [2]:
```
df = pd.read_csv('a4q3.csv', index_col=0)

# Plot the data with each label 'L' getting a different color

#Here we can choose a global colormap, to help visualization
# see https://matplotlib.org/tutorials/colors/colormaps.html
cmapstr = 'Paired'

# here we choose how big each point in the plot will be
ptsize = 2

df.plot.scatter(x='x', y='y',c='L',colormap=cmapstr,s=ptsize, title='True Labels')
plt.show()
```



Step 2

Now we'll strip off the labels, and set up some variables for use by the fitting methods.

In [3]:
```
# just the input features
X_df = df[['x', 'y']]

# just the labels.
L_df = df['L']

# the number of true classes in the data
n_classes = len(np.unique(L_df))

# the number of clusters to seek; experiment with this!
n_components = n_classes
```

Step 3

Fit the KMeans model to the features of the data set.

Step 3 --- Solution

The KMeans object lets us select several options. The most important is the first.

1. `n_clusters` : This is the K in KMeans. We have to choose how many clusters to look for.
    - In our situation, we already know there are 5 labels, so choosing `n_clusters=5` is natural. When the number of clusters is known in advance that's a good choice. The cell above uses the data to count how many classes there are.
    - In some situations, especially if the number of dimensions is high, we might not be able to tell how many clusters we want. We still have to choose!
2. `init` : This lets us choose how to initialize the clusters. There are 2 options in scikit, to use random starting points, and something called `k-means++` .
3. There are others, but these two are the ones we explore in this notebook.

In [4]:
```
from sklearn.cluster import KMeans

# create the model object
kmeans_estimator = KMeans(n_clusters=n_components)

# fit on all the data
kmeans_estimator.fit(X_df)
```

Out[4]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300
,
       n_clusters=5, n_init=10, n_jobs=None, precompute_distances='a
uto',
       random_state=None, tol=0.0001, verbose=0)
```

Step 3 Grading -- 3 marks

1. The number of clusters is required. There's no reason for it to be some value different from 5 or `n_classes`.
2. Later parts of the notebook require changing `init` so either value could appear here. Also, the default value is useful, so the `init` might not be set explicitly.
3. Other options might be chosen, but none are required.

Give full marks here unless the required options are missing.

Step 4

Fit the GMM model to the features of the data set.

Step 4 --- Solution

The GaussianMixture object lets us select several options. The most important is the first.

1. `n_components` : This is the same as K in KMeans, but why it's not called the same is a mystery. We have to choose how many Gaussian components we want in our mixture.
    - In our situation, we already know there are 5 labels, so choosing `n_components=5` is natural.
    - A variant of GMM called BayesianGaussianMixture can be used if the number of clusters is not known in advance.
2. `init_params` : This lets us choose how to initialize the clusters. There are 2 options in scikit, to use random starting points, and `kmeans`.
3. `covariance_type` : Another interesting one, that lets you decide what the covariance matrix will look like. You can force the covariance matrix to be spherical, or to force each cluster to have the same covariance matrix (relative to different means). The default value lets each component have an unrestricted covariance matrix. This would be the only choice, but using one of the alternatives (restricting the covariance calculations) might be less expensive in large data, and less prone to over-fitting when there's too little data.
4. There are others, but these are the ones we explore in this notebook.

In [7]:
```
from sklearn.mixture import GaussianMixture

# create the model object
gmm_estimator = GaussianMixture(n_components=n_components,
                                init_params='kmeans', random_state=245
)

# fit on all the data
gmm_estimator.fit(X_df)
```

Out[7]:
```
GaussianMixture(covariance_type='full', init_params='kmeans', max_it
er=100,
                means_init=None, n_components=5, n_init=1, precision
s_init=None,
                random_state=245, reg_covar=1e-06, tol=0.001, verbos
e=0,
                verbose_interval=10, warm_start=False, weights_init=
None)
```

Step 4 Grading -- 3 marks

1. The number of clusters is required. There's no reason for it to be some value different from 5 or `n_components`.
2. Later parts of the notebook require changing `init_params` so either value could appear here. Also, the default value is useful, so the `init_params` might not be set explicitly.
3. Other options might be chosen, but none are required.

Give full marks here unless the required options are missing.

Step 5

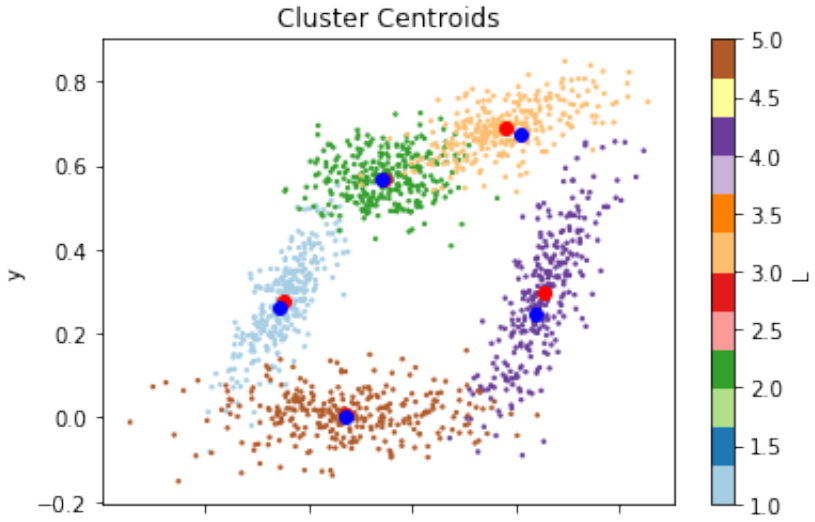
Plot the centroids along with the labelled data.

In [8]:
```
# Plot the data again, using the cluster centers
df.plot.scatter(x='x', y='y',c='L',colormap=cmapstr, s=ptsize, title='
Cluster Centroids')

# plot GMM centroids in red
for centroid in gmm_estimator.means_:
    plt.plot(centroid[0], centroid[1], 'ro')

# plot Kmeans centroids in blue
for centroid in kmeans_estimator.cluster_centers_:
    plt.plot(centroid[0], centroid[1], 'bo')

plt.show()
```



Questions

1. The `KMeans` method allows us to indicate how the initial centroids are chosen:

   - `init='k-means++'` (this is the default behaviour)
   - `init='random'` (you have to ask for this explicitly)

   Using the random initialization, re-run the notebook, and explain the differences that you see. When you're done, return to the default behaviour for the next part.
2. The `GMM` method allows us to indicate how the initial centroids are chosen:

   - `init_params='kmeans'` (this is the default behaviour)
   - `init_params='random'` (you have to ask for this explicitly)

   Using the random initialization, re-run the notebook, and explain the differences that you see.

Answers

1. Switching from default behaviour `init='kmeans++'` to `init='random'` seems to have very little effect. This is of course, dependent on the data set. With other datasets, there could be a difference.
2. Switching from `init_params='kmeans'` to `init_params='random'` has a big effect.

    - With `init_params='kmeans'`, the centroids seem to match the actual clusters that we know are in the dataset, and are fairly close to the kmeans centroids.
    - With `init_params='random'`, the centroids seem to find their home in the middle of all the clusters. I ran the notebook a number of times, but I never saw the random initialization locate the clusters anywhere but the middle.
    - This behaviour is also dependent on the dataset.
        - I suspect that what is happening is that the initial covariances are randomly a lot larger than the covariance of the dataset as a whole, and so never shrinks less than that. Another way to think about it is in terms of scale. If we scaled the sample points by a factor of 10^{-5}, the data would look like a single cluster if we displayed the data to the same scale as in the plot above. At that scale, we'd only see one cluster, and we'd expect a clustering algorithm to fit multiple centroids at the center of it.
        - Another thing that could be going on is that *responsibilities* are assigned randomly, and all the classes have the same number of points. So the centroids would get roughly equal numbers of points from all clusters. The E-step wouldn't be able to move the responsibilities much, because every centroid is equally far from all the points.
        - Without a clearer picture about what the random initial starting point is actually doing, it's not possible to do more than speculate.

Grading

1. For full marks, the comment should note that there is no obvious effect. No explanation is needed.
2. For full marks, the comment should note that using random seems to put the centroids at the same location in the middle of the data. There weas no requirement to explain why it happens.

Grading: 12 marks

- Step 3. 3 marks. Give full marks here unless the required options are missing.
  1. The number of clusters is required. There's no reason for it to be some value different from 5 or `n_classes`.
  2. Later parts of the notebook require changing `init` so either value could appear here. Also, the default value is useful, so the `init` might not be set explicitly.
  3. Other options might be chosen, but none are required.

- Step 4. 3 marks. Give full marks here unless the required options are missing.
  1. The number of clusters is required. There's no reason for it to be some value different from 5 or `n_components`.
  2. Later parts of the notebook require changing `init_params` so either value could appear here. Also, the default value is useful, so the `init_params` might not be set explicitly.
  3. Other options might be chosen, but none are required.

- Answer to question 1 above (3 marks).
  - For full marks, the comment should note that there is no obvious effect. No explanation is needed.

- Answer to question 2 above (3 marks).
  - For full marks, the comment should note that using random seems to put the centroids at the same location in the middle of the data. There was no requirement to explain why it happens.

CMPT 423/820

Assignment 4 Question 4

- Solutions and Grading Scheme
- 12 marks

The `a4q4.csv` file contains a 6D dataset (each sample has 6 numeric features, one class label, and an index column). It's too many dimensions to visualize directly.

To complete this question:

1. Produce a scatter plot of any 2 dimensions. Do you see any natural clusters in 2D?
2. Build a classifier (your choice, e.g., Naive Bayes, KNN, etc) using the 6 features and the label. Get an estimate of the classifier performance using cross validation.
3. Use Principle Component Analysis to reduce the data to 2 principle components. (look at the docs: sklearn.decomposition.PCA)
4. Produce a scatter plot of the 2 principle components. Do you see any natural clusters?
5. Build a classifier (the same type you used above) using the 2 principle components and the label. Get an estimate of the classifier performance using cross validation.
6. Report on the difference in classifier accuracy.

First some libraries...

In [1]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn import decomposition
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import LinearSVC, SVC
from sklearn.neighbors import KNeighborsClassifier
```

Step 1

The `a4q2.csv` file contains a 6D dataset (each sample has 6 numeric features, and one index).

In [2]:
```
df = pd.read_csv('a4q4b.csv', index_col=0)

#Here we can choose a global colormap, to help visualization
# see https://matplotlib.org/tutorials/colors/colormaps.html
cmapstr = 'Paired'

# here we choose how big each point in the plot will be
ptsize = 2

df.plot.scatter(x='x', y='y', c='L',colormap=cmapstr,s=ptsize, title='
True Labels')
plt.show()
```

It's obvious that the data sets overlap, but it's possible to see that there are some patterns: the light blue points are near the lower-left, but the red-brown points are only near the upper right. In 2D, there are no clusters, but one might suspect that in higher dimensions, there may be clusters, if only we could get the right perspective.

Step 2

We'll set up the data set and the labels, so we can apply the various processes.

In [3]:
```
X = df[['x','y','a','b','c','d']]
y = df['L']
```

Step 3: the classifier on the original data

Everyone's favorite: Naive Bayes!

In [4]:
```
clf1 = GaussianNB()
clf2 = GaussianNB()

scores = cross_val_score(clf1, X, y, cv=10)
print("GNB, orig: Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), score
s.std() * 2))
```

```
GNB, orig: Accuracy: 0.92 (+/- 0.05)
```

Accuracy is already pretty good.

Step 4: PCA

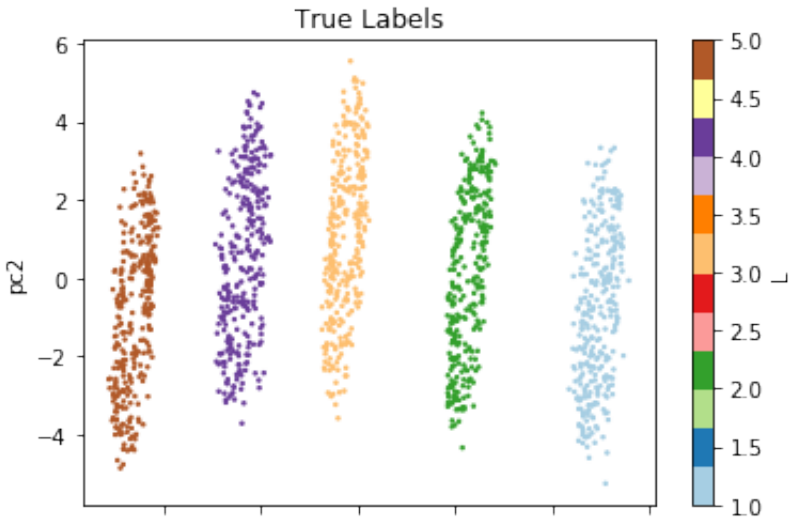
We'll apply PCA, looking for 2 principle components. The primary reason to look for 2 is that we'd like to visualize the data. For any other reason, we might choose a different number of components.

- The Scikit interface use the `fit()` method to do the analysis.
- To project the original 6D data onto the 2 principle components, we use the `transform()` method.
- Scikit provides a convenience method `fit_transform()` that does both in one call.

In [5]:
```
pca = decomposition.PCA(n_components=2)
pca.fit(X)

pca_df = pca.transform(X)

# Now let's see the data projected onto the 2 components.
scatter_df = pd.DataFrame(pca_df,columns=['pc1','pc2'])
scatter_df['L'] = df['L']
scatter_df.plot.scatter(x='pc1', y='pc2', c='L',colormap=cmapstr,s=pts
ize, title='True Labels')
plt.show()
```



This data looks too good to be true, and that's because I designed it to be like that. Five neat clusters. No classifier should have any trouble with this transformed data.

Step 5: classifying the transformed data

Let's see what Naive Bayes has to say about this dataset.

In [6]:
```
scores = cross_val_score(clf2, pca_df, y, cv=10)
print("GNB, 2pca: Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), score
s.std() * 2))
```

```
GNB, 2pca: Accuracy: 1.00 (+/- 0.00)
```

The data is so good that Naive Bayes makes zero error in 10-fold cross-validation. This is a substantial improvement over the accuracy of 90% or so using the 6D dataset.

Grading:

1. 3 marks: You built a classifier using the whole dataset, and reported an accuracy value.
    - Full marks for any classifier and any accuracy results.
        - Naive Bayes: around 90% accuracy.
        - SVM (linear): around 90-100% accuracy.
        - KNN (k=11): around 100% accuracy.

1. 3 marks: You applied PCA, and visualized the first two principle components.

    - Full marks for calling PCA, and plotting the results.
        - Nice neat clusters!
2. 3 marks: You built a classifier using the first two principle components, and reported an accuracy value.

    - Full marks for any classifier and any accuracy results.
        - Naive Bayes: around 100% accuracy.
        - SVM (linear): around 90-100% accuracy.
        - KNN (k=11): around 100% accuracy.
3. 3 marks: You reported on the difference in accuracy between your two classifiers.

    - Full marks for a report that compared the two results, and made some attempt to interpret the results.
        - Naive Bayes: around 100% accuracy. This was a little better than the 6D data. This means that there are useful distinctions in the 6D data that are fairly effectively used by Naive Bayes. The problem is that we can't see them in 6D.
        - SVM (linear): around 90-100% accuracy. The accuracy didn't improve, but it also didn't get worse. That means for this data we could base the classifier on the transformed data, meaning the classifier would be more efficient, and we would not lose any accuracy.
        - KNN (k=11): around 100% accuracy. Using PCA represents an opportunity to build a simpler model using far fewer features, just as described for SVC.

All of these discussions show that the value of PCA depends on the data set, and the model being used for classification.

Appendix -- Using other classifiers

For some perspective, I used several classifiers, just to collect the comments above. Students are not expected or required to use several classifiers.

Support Vector Machines:

In [7]:
```
clf1 = SVC(kernel='linear')
clf2 = SVC(kernel='linear')

scores = cross_val_score(clf1, X, y, cv=10)
print("SVC, orig: Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), score
s.std() * 2))

scores = cross_val_score(clf2, pca_df, y, cv=10)
print("SVC, 2pca: Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), score
s.std() * 2))
```

```
SVC, orig: Accuracy: 1.00 (+/- 0.00)
SVC, 2pca: Accuracy: 1.00 (+/- 0.00)
```

The linear SVM was able to detect the planes that would separate the data. These are not obvious in the visualization of `a4q2b.csv` in any two dimensions, but clearly available by SVM in the 6D space.

In [8]:
```
clf1 = LinearSVC(max_iter=5000)
clf2 = LinearSVC(max_iter=5000)

scores = cross_val_score(clf1, X, y, cv=10)
print("LinearSVC, orig: Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(),
scores.std() * 2))

scores = cross_val_score(clf2, pca_df, y, cv=10)
print("LinearSVC, 2pca: Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(),
scores.std() * 2))
```

```
LinearSVC, orig: Accuracy: 0.95 (+/- 0.03)
LinearSVC, 2pca: Accuracy: 0.90 (+/- 0.04)
```

The alternate implementation doesn't do as well as the more general SVC implementation, as a result of the specialization. Not exactly sure why! The PCA actually caused this version to do worse than the original 6D dataset.

K Nearest Neighbours

Because Naive Bayes showed some change, and SVC was more or less the same, I wanted another vote on the matter.

In [9]:
```
clf1 = KNeighborsClassifier(11)
clf2 = KNeighborsClassifier(11)

scores = cross_val_score(clf1, X, y, cv=5)
print("KNN, orig: Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), score
s.std() * 2))

scores = cross_val_score(clf2, pca_df, y, cv=5)
print("KNN, 2pca: Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), score
s.std() * 2))
```

```
KNN, orig: Accuracy: 1.00 (+/- 0.00)
KNN, 2pca: Accuracy: 1.00 (+/- 0.00)
```

KNN does equally well on 6D as on the 2 principle components from PCA. This supports the idea that the `a4q2b.csv` data is well-separated in 6D even though we can't visualize it. Using PCA makes the classifier simpler when all is said and done. Because KNN stores data internally, storing 2D data is much better than storing 6D data, by a factor of 3.